Răzvan Rughiniș
Răzvan Deaconescu

George Milescu
Mircea Bardac

Computer Science

Introducere în sisteme de operare

Editura PRINTECH

Material protejat de autor

Cuvânt înainte

Noțiunea de sistem de operare reprezintă, probabil, unul dintre termenii cei mai des întâlniți în domeniul calculatoarelor, și nu numai. De la formele greoaie dezvoltate în anii '60, cunoscute doar profesioniștilor, sistemele de operare au cunoscut o transformare continuă, strâns corelată cu dezvoltarea sistemelor de calcul și a tehnologiilor asociate. În ziua de astăzi, sistemele de operare oferă o interfață facilă și prietenoasă atât utilizatorilor obișnuiți ai serviciilor Internet, cât și utilizatorilor comerciali ai aplicațiilor dedicate, celor ce folosesc facilitățile multimedia și jocuri, sau celor profesioniști care dezvoltă aplicații sau întrețin sisteme de calcul și rețele de calculatoare. Evoluția tehnologică a dus la dezvoltarea sistemelor de operare pentru un număr tot mai mare de dispozitive, de la sisteme server, desktop și laptop la PDA-uri și smartphone-uri.

Cartea de față își propune familiarizarea cititorului cu lumea sistemelor de operare și, în particular, cu latura preponderent tehnică a acestora. Am creat această lucrare având în permanență în vedere cunoștințele de bază și cadrul conceptual necesare unui student la o facultate de calculatoare. În această structură, cartea este însă construită pentru a fi utilă oricărui cititor care caută un prim contact cu domeniul sistemelor de operare. Sperăm ca parcurgerea sa să ofere și un set de deprinderi și abordări în soluționarea problemelor care depășesc sfera sistemelor de operare.

Diversitatea subiectelor abordate a reprezentat o dificultate în crearea unei succesiuni clare de capitole. Strategia aleasă este una stratificată, fiecare capitol bazându-se pe cele studiate anterior. Totuși, au existat momente în care a trebuit să utilizăm anumite noțiuni înainte de a fi definit cadrul conceptual, sau la câteva capitole distanță de prezentarea lor. În astfel de situații cititorului îi sunt oferite referințe către capitolele în care sunt clarificate noțiunile invocate.

Cartea urmărește prezentarea și discutarea noțiunilor de bază necesare unui student în primii ani de facultate, în domeniul calculatoarelor. Diversitatea subiectelor și nivelul de detaliu recomandă o asimilare în profunzime a informațiilor, dincolo de durata unui semestru sau a unui an. Sperăm ca studentul dornic de aprofundare să răsfoiască această carte în momentele în care caută sprijin suplimentar pentru rezolvarea unei probleme din domeniu.

Din punct de vedere tehnic, materialul de față oferă o perspectivă ce aparține preponderent universului Linux. Am considerat contactul cu Linux ca pe o oportunitate aparte pentru o majoritate a utilizatorilor ce provin din mediul Windows, în care deseori alternativele în domeniul sistemelor de operare nu reprezintă o opțiune luată în considerare. Dorința noastră este ca utilizarea unui nou sistem de operare, cu o răspândire și o evoluție tot mai intense, să ofere o nouă perspectivă asupra lumii calculatoarelor în general și a sistemelor de operare în particular. Deși cartea este

focalizată pe Linux, fiecare capitol include secțiuni de studii de caz în care sunt prezentate mecanismele similare dintr-un sistem Windows.

Structura cărții este concepută pentru a oferi atât o prezentare a cadrului conceptual, cât și o parte aplicativă construită prin exemple. Fiecare capitol este prefațat de o mică secțiune „Ce se învață în acest capitol?", utilă pentru reperarea principalelor noțiuni. Capitolele se încheie cu o secțiune de „Cuvinte cheie" și apoi de „Întrebări", pentru a permite cititorului o autoevaluare a cunoștințelor dobândite. Unele secțiuni sunt marcate cu simboluri grafice cu o semnificație specială: important, notă, OZN (pentru tehnologiile recente de tipul „bleeding edge") și atom (pentru aspecte tehnice avansate).

Recomandăm parcurgerea secvențială a cărții, dar cititorul avansat poate sări direct la un capitol de interes particular. Dat fiind conținutul practic detaliat, este utilă folosirea calculatorului pentru rularea comenzilor prezentate și pentru explorarea opțiunilor existente, în paralel cu parcurgerea noțiunilor teoretice.

Mulțumim tuturor celor care au contribuit la realizarea cărții. Modul de organizare și prezentare, ca și diversitatea informațiilor prezentate se bazează pe efortul continuu și pasiunea unei echipe entuziaste. În primul rând, îi mulțumim domnului profesor Nicolae Ţăpuș, precum și colegilor noștri Vlad Dogaru, Mihai Maruseac, Daniel Rosner și Andrei Buhaiu, a căror implicare a constituit un beneficiu direct în elaborarea acestei cărți. Mulțumim, de asemenea, colegilor Alex Eftimie și Andrei Faur pentru contribuția adusă, și colegilor Alex Juncu, Lucian Grijincu, Călin Iorgulescu, Voichița Iancu, Andrei Dumitru, Laura Gheorghe pentru revizuirea materialului pe parcursul finalizării acestuia.

Adresăm mulțumiri speciale echipei cursului de Utilizarea Sistemelor de Operare care ne-a oferit o atmosferă de suport, implicare și energie pentru realizarea cărții. Forma actuală a cărții se bazează pe efortul susținut depus de-a lungul numeroaselor activități din jurul cursului de Utilizarea Sistemelor de Operare.

Nu în ultimul rând, mulțumim cititorilor, la primul pas în domeniul plin de provocări și de satisfacții al calculatoarelor. Ei sunt cei cărora le dedicăm această carte. Ne-a făcut plăcere să o scriem, și sperăm că măcar o parte din entuziasmul nostru să se convertească în pasiune pentru cititorii acestor pagini.

Autorii

Cuprins

1 Introducere **1**

1.1 Ce este un sistem de operare? . . . 1

1.1.1 Funcțiile unui sistem de operare . . . 3

1.1.2 Tipuri de sisteme de operare . . . 4

1.1.3 Functionarea sistemelor de operare . . . 5

1.2 Scurt istoric al sistemelor de operare . . . 7

1.2.1 Prima generație . . . 7

1.2.2 A doua generație . . . 8

1.2.3 A treia generatie . . . 8

1.2.4 A patra generație . . . 9

1.3 Sisteme de operare moderne . . . 9

1.3.1 Sisteme de operare desktop și server . . . 10

1.3.2 Familia Windows . . . 11

1.3.3 Mac OS X . . . 11

1.3.4 Linux . . . 12

1.3.5 Alte Unix-uri . . . 13

1.3.6 Comunități open source . . . 14

1.3.7 Sisteme de operare pentru dispozitive mobile . . . 15

1.4 Studiu de caz . . . 16

1.4.1 Virtualizare. Rularea unui SO dintr-o mașină virtuală . . . 16

2 Instalarea Linux. Configurări de bază **23**

2.1 Linux. Distribuții Linux . . . 23

2.1.1 Principalele distribuții Linux . . . 24

2.1.2 Debian. Ubuntu . . . 25

2.1.3 Tipuri de distribuții Linux . . . 27

2.2 Instalarea Linux . . . 28

2.2.1 Alegerea distribuției Linux . . . 28

2.2.2 Pregătirea sistemului . . . 29

2.2.3 Live CD . . . 31

2.2.4 Instalare Kubuntu . . . 32

2.3 Interacțiunea cu sistemul de operare . . . 38

2.3.1 Interfețele cu utilizatorul . . . 38

2.3.2 Interfața în linie de comandă (CLI) . . . 40

2.3.3 Oprirea sistemului de calcul . . . 44

2.4 Configurări de bază ale SO . . . 45

2.4.1 Administrarea sistemului Linux . . . 45

2.4.2 Asigurarea conectivității la Internet . . . 47

2.4.3 Actualizarea sistemului și a pachetelor . . . 47

2.4.4 Administrarea utilizatorilor și a grupurilor de utilizatori . . . 48

2.5 Studii de caz 48
2.5.1 Interoperabilitatea sistemului de fișiere între Linux și Windows . 48
2.5.2 GParted 49

3 Gestiunea pachetelor și utilizatorilor 53
3.1 Gestiunea utilizatorilor 53
3.1.1 UID, GID 55
3.1.2 Adăugarea și ștergerea utilizatorilor 55
3.1.3 Adăugarea și ștergerea unui grup de utilizatori 56
3.1.4 Modificarea datelor unui utilizator 56
3.1.5 Adăugarea și ștergerea utilizatorilor 57
3.2 Gestiunea pachetelor 58
3.3 Studii de caz 64
3.3.1 Fișierele în care sunt stocate informații despre utilizatori 64
3.3.2 Actualizarea unui sistem Debian sau Ubuntu 66

4 Sisteme de fișiere 69
4.1 Noțiuni introductive 69
4.1.1 Ce este un sistem de fișiere 69
4.1.2 Ierarhia sistemului de fișiere 70
4.1.3 Căi relative și căi absolute 72
4.1.4 Variabila de mediu PATH 74
4.2 Tipuri de fișiere 74
4.2.1 Terminologie 74
4.2.2 Detectia tipului fișierelor 75
4.3 Operații uzuale asupra fișierelor și directoarelor 76
4.3.1 Afișarea și schimbarea directorului curent 76
4.3.2 Afișarea conținutului fișierelor 77
4.3.3 Listarea conținutului unui director 77
4.3.4 Crearea fișierelor/directoarelor 79
4.3.5 Copiere/mutare/redenumire/ștergere 80
4.3.6 Arhivarea fișierelor și dezarhivarea 82
4.3.7 Backup 85
4.4 Redirectări de comenzi 85
4.4.1 Descriptorii de fișier 85
4.4.2 Redirectări 86
4.5 Drepturi de acces 87
4.5.1 Utilizatori și grupuri de utilizatori vs. liste de acces 88
4.5.2 Clasificarea drepturilor de acces 89
4.5.3 Vizualizarea drepturilor de acces 89
4.6 Căutarea fișierelor 90
4.6.1 Comanda find 90
4.6.2 Comanda locate 91
4.6.3 Comanda whereis 91
4.6.4 Comanda which 92
4.6.5 Comanda type 92
4.7 Tipuri de sisteme de fișiere 92
4.7.1 Integritatea datelor 93
4.7.2 Alegerea unui sistem de fișiere 93
4.7.3 Adresarea într-un sistem de fișiere 95

4.8 Lucrul cu sistemele de fișiere . . . 95
4.8.1 Crearea unui sistem de fișiere . . . 96
4.8.2 Montarea unui sistem de fișiere . . . 97
4.8.3 Repararea unui sistem de fișiere . . . 100
4.8.4 Crearea unei imagini pentru un sistem de fișiere . . . 100
4.8.5 "Ştergerea" unui sistem de fișiere . . . 101
4.8.6 Monitorizarea utilizării discului . . . 101
4.9 Tendințe în sistemele de fișiere . . . 102
4.9.1 Sisteme de fișiere în userspace . . . 102
4.9.2 ZFS, ext4, btrfs . . . 104
4.10 Studii de caz . . . 104
4.10.1 Comenzi pentru lucrul cu fișiere în Windows . . . 104
4.10.2 Utilizarea în siguranță a comenzilor pe fișiere . . . 104
4.10.3 Montarea unui sistem de fișiere FAT32 astfel încât toți utilizatorii să aibă drept de scriere pe el . . . 105
4.10.4 Montarea unui sistem de fișiere FAT32 astfel încât utilizatorii dintr-un anumit grup să aibă drept de scriere pe acesta . . . 106
4.10.5 Sistem de fișiere într-un fișier . . . 106
4.10.6 ntfs-3g . . . 107

5 Procese 111
5.1 Noțiuni introductive . . . 111
5.1.1 Ce este un proces? . . . 112
5.1.2 Deosebirea dintre un proces și un program . . . 112
5.1.3 Structura unui proces . . . 113
5.1.4 Stările unui proces. Multiprogramare și multiprocesare . . . 115
5.2 Ierarhia de procese în Unix. Vizualizarea proceselor sistemului . . . 116
5.2.1 Utilitarul ps . . . 117
5.2.2 Utilitarul pstree . . . 122
5.2.3 Utilitarul pgrep . . . 123
5.2.4 Utilitarul top . . . 124
5.2.5 Timpul de execuție al unui proces. Comanda time . . . 126
5.2.6 Sistemul de fișiere procfs . . . 127
5.3 Rularea proceselor în background. Job-uri. Daemoni . . . 130
5.3.1 Rularea unui proces în background . . . 130
5.3.2 Suspendarea unui proces . . . 131
5.3.3 Controlul job-urilor . . . 131
5.3.4 Daemoni . . . 133
5.4 Semnale . . . 135
5.4.1 Semnale importante UNIX . . . 136
5.4.2 Comenzile kill, killall și pkill . . . 137
5.4.3 Transmiterea de semnale prin combinații de taste . . . 139
5.5 Comunicația între procese . . . 139
5.5.1 Operatorul | (pipe) . . . 140
5.6 Swapping . . . 141
5.7 Studii de caz . . . 142
5.7.1 Managementul proceselor/serviciilor pe Windows . . . 142
5.7.2 Procese importante . . . 143
5.7.3 Prioritatea unui proces . . . 145

6 Pornirea și inițializarea sistemului 151
6.1 Pornirea sistemului . . . 151
6.1.1 Problematica pornirii sistemului – bootstrapping . . . 152
6.1.2 Etapele pornirii sistemului . . . 152
6.1.3 BIOS . . . 153
6.1.4 POST . . . 154
6.2 Bootloader . . . 154
6.2.1 Dispozitive boot-abile . . . 155
6.2.2 Structura sectorului de boot pentru un dispozitiv boot-abil . . . 156
6.2.3 Mecanismul de funcționare a unui bootloader . . . 158
6.2.4 GRUB . . . 159
6.3 Configurarea GRUB . . . 162
6.4 Încărcarea nucleului . . . 165
6.4.1 Imaginea de nucleu . . . 165
6.4.2 Opțiuni de boot-are pentru nucleu . . . 166
6.5 Inițializarea sistemului . . . 167
6.5.1 init . . . 167
6.5.2 upstart . . . 171
6.5.3 getty și login . . . 173
6.5.4 Sesiune de shell . . . 174
6.6 Studiu de caz . . . 175
6.6.1 Grub 2 . . . 175
6.6.2 Pornirea sistemului de operare Windows XP . . . 177
6.6.3 Interoperabilitate Linux-Windows . . . 179
6.6.4 Personalizarea ecranului GRUB . . . 180

7 Analiza hardware a sistemului 185
7.1 Structura unui sistem de operare . . . 185
7.2 Considerente hardware . . . 186
7.2.1 Structura unui sistem de calcul . . . 187
7.2.2 Procesorul . . . 188
7.2.3 Ierarhia de memorie . . . 188
7.2.4 Placa de bază; interconectarea componentelor . . . 191
7.2.5 Dispozitive periferice; magistrale . . . 192
7.3 Suportul pentru dispozitive la nivelul kernel-ului . . . 193
7.3.1 Listarea modulelor încărcate la un moment dat în sistem . . . 194
7.3.2 Încărcarea unui modul . . . 195
7.3.3 Descărcarea unui modul din kernel . . . 195
7.4 Analiza hardware a unui sistem (magistrale, chipset, CPU, memorie, dispozitive) . . . 196
7.4.1 Lista hardware – /sys . . . 196
7.4.2 Comenzi pentru afișarea dispozitivelor . . . 196
7.4.3 Monitorizarea stării dispozitivelor hardware . . . 199
7.5 Interfața cu dispozitivele din userspace – **udev** și `/dev` . . . 201
7.6 Lucrul cu dispozitive (`/dev`) . . . 202
7.6.1 Tipuri de dispozitive . . . 203
7.6.2 Întreruperi hardware . . . 204
7.6.3 Adrese I/O . . . 205
7.6.4 Adrese DMA . . . 206

7.6.5 Manipularea datelor la nivel scăzut (comanda dd) 206
7.7 Analiza sistemului 208
7.7.1 Informații despre sistem 208
7.7.2 Kernel "tunables" 210
7.8 Studii de caz 211
7.8.1 Salvarea și restaurarea MBR și a tabelei extinse de partiții . . . 211
7.8.2 Salvarea conținutului unui disc cu sectoare inaccesibile 212
7.8.3 Crearea unei imagini de CD; montarea unei imagini 212
7.8.4 Utilizarea unui fișier de pe o partiție FAT32 ca fișier de swap pentru un LiveCD Linux 213

8 Configurări de rețea 217
8.1 Concepte de rețea 217
8.1.1 Noțiuni de bază 218
8.1.2 IPv6 219
8.1.3 Moduri de adresare: adresare unicast, multicast, broadcast . . 219
8.2 Parametri de rețea 220
8.2.1 Adresă IP și mască de rețea 220
8.2.2 Ruter implicit (default gateway) 222
8.2.3 DNS 223
8.3 Configurări temporare 225
8.3.1 Interfețe de rețea. Configurări permanente și configurări temporare 225
8.3.2 Configurarea temporară statică a unei adrese IP pe o interfață . 226
8.3.3 Asigurarea conectivității la Internet. Configurarea temporară statică a unei rute implicite 229
8.4 Configurări permanente 230
8.4.1 Configurarea permanentă a unei interfețe de rețea 230
8.4.2 Configurarea permanentă statică a unei rute implicite 232
8.4.3 DHCP. Configurarea unei interfețe în mod automat 232
8.4.4 DNS. Configurarea serverelor de DNS 233
8.4.5 Aliasuri. /etc/hosts 235
8.4.6 Configurarea numelui stației curente 235
8.5 Testarea configurațiilor de rețea 236
8.5.1 Ping 236
8.5.2 Traceroute 237
8.5.3 Cum se depistează problemele uzuale în cazul configurărilor de rețea 238
8.6 Studii de caz 240
8.6.1 Verificarea în linie de comandă a parametrilor de rețea în Windows 240
8.6.2 Configurări de rețea în Windows Vista 241
8.6.3 Configurarea PPPoE în Linux 243

9 Servicii de rețea 249
9.1 Concepte specifice aplicațiilor de rețea 249
9.1.1 Stiva TCP/IP 249
9.1.2 Implementarea parțială a stivei TCP/IP 252
9.1.3 Modelul client-server 254
9.1.4 Porturi 255

9.2 Execuția comenzilor la distanță . . . 257
9.2.1 Telnet . . . 257
9.2.2 SSH . . . 258
9.2.3 Utilitarul wget . . . 262
9.3 Email – Poșta electronică . . . 263
9.3.1 Arhitectură și funcționare . . . 264
9.3.2 Clienți de email . . . 266
9.3.3 Securitatea serviciului de email . . . 267
9.4 WWW . . . 268
9.4.1 Tehnologiile de bază ale WWW . . . 269
9.4.2 Funcționarea serviciului . . . 272
9.4.3 Servere Web . . . 272
9.4.4 Clienții Web . . . 273
9.5 Studii de caz . . . 273
9.5.1 Utilitarul cURL . . . 273
9.5.2 FTP . . . 274

10 Elemente de securitate 279
10.1 Problematica securității . . . 279
10.1.1 Principii de bază . . . 280
10.1.2 Termeni . . . 283
10.2 Securizarea sistemului . . . 284
10.2.1 Securitatea sistemului de operare . . . 284
10.2.2 Controlul accesului . . . 286
10.2.3 Parole . . . 287
10.2.4 Securitatea sistemului de fișiere . . . 289
10.3 Întreținerea sistemului . . . 293
10.3.1 Monitorizarea sistemului . . . 293
10.3.2 Jurnalizarea și gestiunea jurnalelor . . . 296
10.3.3 Limitarea drepturilor . . . 298
10.4 Atacuri de rețea . . . 303
10.4.1 Tipuri de atacuri în rețea . . . 303
10.4.2 Viruși, viermi, troieni . . . 305
10.4.3 Scanarea porturilor . . . 307
10.5 Securizarea rețelei . . . 309
10.5.1 Firewall-uri . . . 310
10.5.2 Criptarea informației . . . 310
10.5.3 Monitorizarea rețelei . . . 312
10.6 Studiu de caz . . . 314
10.6.1 Drepturile pe fișiere în NTFS . . . 314
10.6.2 Recuperarea parolei . . . 315

11 Compilare și linking 319
11.1 Introducere . . . 319
11.1.1 Editoare . . . 320
11.1.2 Compilare și interpretare . . . 321
11.1.3 De la sursă la executabil, de la executabil la proces . . . 322
11.1.4 Pachete necesare . . . 323
11.2 Compilare. GCC . . . 324
11.2.1 Utilizare GCC . . . 325

11.2.2 Compilarea din surse multiple . . . 327
11.3 Etapele compilării (inclusiv link-editarea) . . . 328
11.3.1 Preprocesarea . . . 330
11.3.2 Compilarea . . . 331
11.3.3 Asamblarea . . . 333
11.3.4 Optimizarea compilării . . . 335
11.3.5 Link-editarea . . . 337
11.3.6 Fișiere executabile . . . 339
11.4 Biblioteci de funcții . . . 341
11.4.1 Tipuri de biblioteci . . . 342
11.4.2 Informații despre bibliotecile de funcții . . . 343
11.4.3 Utilizarea bibliotecilor . . . 344
11.5 Automatizarea sarcinilor – make . . . 345
11.5.1 Cel mai simplu Makefile . . . 346
11.5.2 Folosirea dependențelor . . . 347
11.5.3 Dependențe ierarhice . . . 347
11.5.4 Target-ul clean . . . 350
11.5.5 Target-urile .PHONY și all . . . 350
11.5.6 Variabile în Makefile . . . 352
11.5.7 Sintaxă Makefile . . . 352
11.5.8 Moduri de utilizare a Make . . . 353
11.6 Portabilitate . . . 354
11.6.1 Portabilitatea la nivelul arhitecturii sistemului de calcul . . . 354
11.6.2 Portabilitatea unui limbaj de programare . . . 354
11.6.3 Portabilitatea la nivelul sistemului de operare . . . 356
11.7 Studiu de caz . . . 356
11.7.1 GCC în Windows . . . 356
11.7.2 Link-editarea modulelor C și a modulelor C++ . . . 357

12 Shell scripting 365
12.1 Noțiuni introductive . . . 365
12.1.1 De ce shell scripting? . . . 366
12.1.2 Facilități oferite de scripturile shell . . . 367
12.2 Interacțiunea cu shell-ul . . . 368
12.2.1 Editarea comenzilor . . . 368
12.2.2 Folosirea istoricului. Completare automată . . . 369
12.2.3 Comenzi interne (*built-in*) și comenzi externe . . . 371
12.2.4 Execuția unei comenzi shell . . . 373
12.3 Scripturi shell . . . 374
12.3.1 Cel mai simplu script shell . . . 374
12.3.2 Comentarii într-un script shell . . . 376
12.3.3 Comenzi simple shell . . . 376
12.3.4 One liners . . . 378
12.4 Programarea shell . . . 384
12.4.1 Variabile . . . 384
12.4.2 Caractere speciale shell . . . 387
12.4.3 Instrucțiuni de decizie . . . 391
12.4.4 Cicluri în shell . . . 395
12.5 Filtre de text . . . 399

12.5.1 cat, tac, nl . . . 400
12.5.2 sort, uniq . . . 400
12.5.3 head, tail . . . 402
12.5.4 cut . . . 403
12.5.5 tr . . . 404
12.5.6 wc . . . 405
12.5.7 grep . . . 405
12.5.8 sed . . . 408
12.5.9 awk . . . 410
12.6 Comenzi de lucru cu fișiere . . . 414
12.6.1 xargs . . . 414
12.6.2 locate . . . 415
12.6.3 find . . . 416
12.7 Expandarea în shell . . . 419
12.7.1 Simbolul $ . . . 419
12.7.2 Expresii regulate în shell . . . 421
12.8 Parametrii unui script shell . . . 421
12.8.1 Comanda shift . . . 422
12.8.2 Parametri speciali . . . 423
12.8.3 Exemplu de utilizare a parametrilor . . . 423
12.9 Funcții . . . 424
12.9.1 Sintaxa unei funcții . . . 424
12.9.2 Parametrii unei funcții . . . 425
12.10 Scripturile de pornire Bash . . . 426
12.10.1 Variabile de mediu . . . 427
12.11 Studii de caz . . . 428
12.11.1 Contorizarea numărului de utilizatori autentificați în sistem . . . 428
12.11.2 Schimbarea promptului shell . . . 429
12.11.3 Batch scripting în Windows . . . 431
12.11.4 PowerShell pe Windows . . . 433

13 Mediul grafic 441
13.1 Concepte în mediul grafic . . . 442
13.1.1 Tipuri de imagini . . . 442
13.1.2 Fonturi. Unicode . . . 443
13.2 Interfața grafică în Linux. Componente . . . 444
13.2.1 X Window System . . . 444
13.2.2 Arhitectura X Window System . . . 445
13.3 Pornirea și oprirea interfeței grafice . . . 448
13.4 Configurarea serverului X . . . 450
13.4.1 Configurarea rezoluției . . . 450
13.4.2 Configurarea tastaturii . . . 451
13.5 Configurarea sistemului din KDE . . . 452
13.5.1 System Settings . . . 452
13.5.2 Schimbarea aspectului interfeței grafice . . . 452
13.5.3 Configurări de bază ale sistemului de operare . . . 453
13.5.4 Configurări administrative . . . 454
13.5.5 Configurarea rețelei . . . 454
13.5.6 Managementul utilizatorilor . . . 454

13.5.7 Monitorizarea sistemului și managementul proceselor 455
13.5.8 Gestiunea pachetelor 457
13.6 Servicii desktop 458
13.6.1 Pornirea facilă a aplicațiilor 458
13.6.2 Căutarea fișierelor în sistem 459
13.6.3 Notificări 460
13.7 Aplicații KDE vs. GNOME 460
13.8 Studii de caz 460
13.8.1 Configurarea X peste SSH 460
13.8.2 Instalarea și configurarea VNC 462
13.8.3 Remote Desktop Connection 464

14 Utilitare pentru dezvoltare 469
14.1 Introducere 469
14.2 Coding style (indent, astyle) 470
14.3 Editorul Vim 473
14.4 Sisteme de control al versiunii 479
14.4.1 Principii 479
14.4.2 Subversion 480
14.4.3 Git 481
14.5 Analiza și parcurgerea codului 481
14.5.1 ctags 482
14.5.2 Analiza statică a codului – splint 482
14.6 Automatizarea compilării 484
14.6.1 Compilarea unei aplicații din surse 484
14.7 Execuția unui program 485
14.7.1 Zonele de memorie ale unui executabil și proces 485
14.7.2 Utilizarea bibliotecilor partajate 486
14.7.3 Analiza apelurilor de sistem și a semnalelor 487
14.8 Depanarea unui program 489
14.8.1 gdb, ddd 489
14.8.2 valgrind 492
14.9 Medii integrate de dezvoltare 494
14.9.1 Eclipse 494
14.9.2 Anjuta 494
14.10 Managementul proiectelor software 496
14.10.1 Procese de dezvoltare software 496
14.10.2 Aplicații web pentru managementul proiectelor software 497
14.10.3 Resurse online pentru dezvoltarea proiectelor 498
14.11 Resurse de documentare pentru dezvoltator 498
14.11.1 Documentație oficială 498
14.11.2 Documentarea programelor proprii 499
14.11.3 Cărți și tutoriale 500
14.11.4 Documentație din Internet 501
14.12 Studiu de caz 502
14.12.1 Microsoft Visual Studio 502

15 Viața în Linux 507
15.1 Muzica în Linux 507
15.2 Filme în Linux 509

15.3 Scrierea unui CD/DVD în Linux 510
15.4 Messenger în Linux 511
15.5 BitTorrent în Linux 512
15.6 Imprimanta în Linux 514
15.7 Jocuri în Linux 515
15.8 Dual-monitor în Linux 516
15.9 Documente office în Linux 518

A Răspunsuri la întrebări **521**

Bibliografie **529**

Glosar **531**

Abrevieri

ACL – Access Control List
ACPI – Advanced Configuration and Power Interface
AGP – Advanced Graphics Port
API – Application Programming Interface
BIOS – Basic Input Output System
CLI – Command Line Interface
CMOS – Complementary Metal Oxide Semiconductor
CPU – Central Processing Unit
DDD – Data Display Debugger
DE – Desktop Environment
DHCP – Dynamic Host Configuration Protocol
DLL – Dynamic-link library
DM – Display Manager
DMA – Direct Memory Access
DNS – Domain Name System
DoS – Denial of Service
DRAM – Dynamic RAM
ELF – Executable and Linking Format
FSB – Front Side Bus
FTP – File Transfer Protocol
FUSE – Filesystem in Userspace
GCC – GNU Compiler Collection
GDB – GNU Debugger
GID – Group Identifier
GNU – GNU's Not Unix
GNU CPP – GNU C preprocessor
GUI – Graphical User Interface
HDD – hard disk drive
HTML – Hypertext Markup Language
HTTP – Hypertext Transfer Protocol
ICH – I/O Controller Hub
ICMP – Internet Control Message Protocol
IDE – Integrated Development Environment
IDS – Intrusion Detection System
IMAP – Internet Mail Access Protocol
IP – Internet Protocol
IRC – Internet Relay Chat
IRQ – interrupt request
ISP – Internet Service Provider

LAN – Local Area Network
LTS – Long Time Support
LVM – Logical Volume Manager
MAN – Metropolitan Area Network
MBR – Master Boot Record
MCH – Memory Controller Hub
MIME – Multipuprose Internet Mail Extensions
MinGW – Minimalist GNU for Windows
MSVC – Microsoft Visual C++
NAT – Network Address Translation
PCI – Peripheral Component Interconnection
PID – process id
PKI – Public Key Infrastructure
POP3 – Post Office Protocol
POSIX – Portable Operating System Interface
POST – Power-on Self Test
PPP – Point to Point Protocol
PPPoE – PPP over Ethernet
RAID – Redundant Array of Inexpensive Disks
RAM – Random Access Memory
RDC – Remote Desktop Connection
RPM – Rotations Per Minute
RTOS – sisteme de operare în timp real
S.M.A.R.T. – Self-Monitoring, Analysis and Reporting Technology
SMTP – Simple Mail Transfer Protocol
SNMP – Simple Network Management Protocol
SO – sistem de operare
SRAM – Static RAM
SSH – Secure Shell
SVN – Subversion
TCP – Transmission Control Protocol
TFTP – Trivial File Transfer Protocol
TLD – Top Level Domain
TTL – Time To Live
TUI – Text User Interface
UDP – User Datagram Protocol
UID – User Identifier
URL – Uniform Resource Locator
USB – Universal Serial Bus
VBR – Volume Boot Record
VCS – Version Control System
VNC – Virtual Network Computing
WAN – Wide Area Network
WIMP – window, icon, menu, pointing device
WLAN – Wireless LAN
WM – Window Manager
WWW – World Wide Web
XHTML – Extensible Hypertext Markup Language

Capitolul 1

Introducere

They have computers, and they may have other weapons of mass destruction.

Janet Reno

Ce se învață din acest capitol?

- Ce este un sistem de operare
- Funcțiile unui sistem de operare
- Tipuri de sisteme de operare
- Istoricul sistemelor de operare
- Sisteme de operare moderne
- Windows, Mac OS X, Linux
- Virtualizare

1.1 Ce este un sistem de operare?

Sistemele de operare reprezintă un concept familiar tuturor utilizatorilor de calculatoare personale. Ele apar tot mai des și în implementarea de funcționalități din alte aparate, de la telefoanele mobile la cardurile bancare inteligente. Există însă și dispozitive care nu au (încă) un sistem de operare, în principal deoarece execută un număr mic de operațiuni care nu necesită nicio modificare. De exemplu, calculatorul dintr-un cuptor cu microunde execută un program simplu fixat prin circuite fizice. Sistemele de operare sunt utile în cazul calculatoarelor care trebuie să se adapteze unui mediu în schimbare, ele permițând modificarea comportamentului calculatorului fără schimbarea succesiunii operațiilor hardware.

Un **sistem de operare** (abreviat în continuare *SO*) este un set de programe care controlează distribuția resurselor unui calculator și mediază comunicarea dintre hardware, pe de o parte, și aplicațiile utilizatorilor, pe de altă parte.

Scopul unui SO este să asigure stabilitate și flexibilitate în funcționarea calculatorului. Sistemele de operare permit, în general, interacțiunea cu mai mulți utilizatori, rularea mai multor aplicații, și modificarea strategiilor de răspuns ale calculatorului la o anumită problemă.

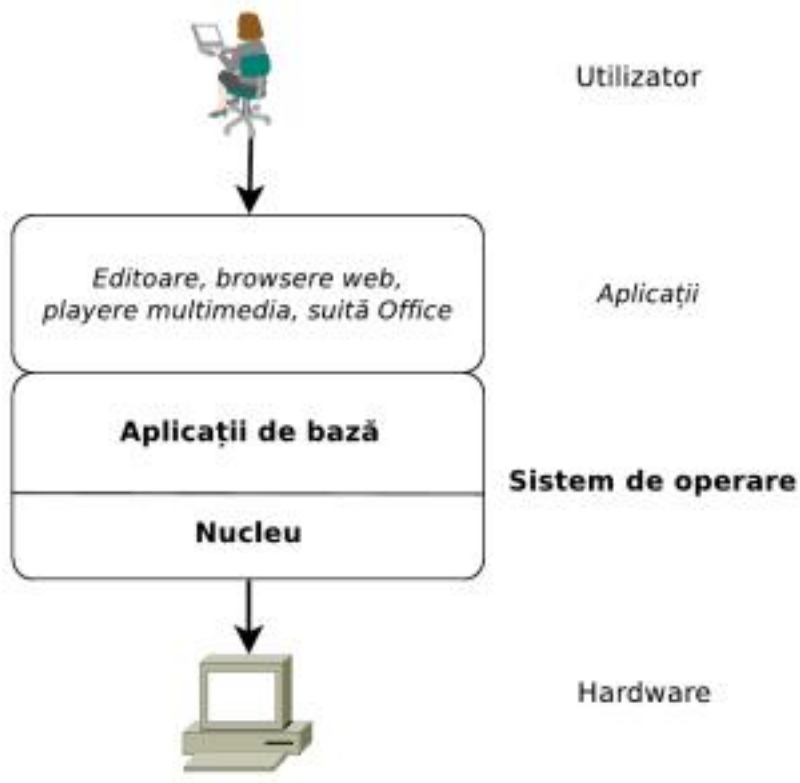


Figura 1.1: Rolul unui sistem de operare

În figura 1.1 este prezentat rolul de mediere al sistemului de operare între utilizatori și hardware. Sistemul de operare va avea acces direct la resursele hardware, în vreme ce utilizatorii folosesc cel mai adesea aplicații ce apelează prin intermediul bibliotecilor sau direct funcții ale sistemului de operare.

Componenta esențială a sistemului de operare o reprezintă **nucleul** sau **kernelul**. Acesta are rolul de gestiune a resurselor fizice și de oferire a unei interfețe comune aplicațiilor. Nucleul nu este o aplicație în sine, ci este un mediu în cadrul căruia pot rula alte aplicații.

Programele de bază sunt aplicațiile fundamentale ale unui sistem de operare care permit interacțiunea cu nucleul și cu sistemul fizic.

Astfel de aplicații sunt interpretoarele de comenzi, utilitare de gestiune a utilizatorilor și a sistemului de fișiere, biblioteci etc. În general nu este bine conturată definiția de aplicație de bază; se consideră că o aplicație este o aplicație de bază atunci când oferă servicii pentru o bună parte din celelalte componente ale sistemului.

Din punct de vedere *strict tehnic* sistemul de operare este însuși nucleul. Din perspectiva utilizatorului sitemul de operare este compus din nucleu și suita programelor de bază. Programele de bază sunt acele aplicații ce permit utilizatorului interacțiunea cu resursele hardware ale sistemului, această relație fiind ilustrată în figura 1.1.

Comparativ, un calculator poate fi privit ca un aeroport, iar sistemul de operare ca turnul de control care coordonează traficul. Resursele aeroportului sunt reprezentate de pistele și avioanele pe care le pune la dispoziție. Fără un turn de control care să asigure gestiunea și organizarea acestor resurse, pe un aeroport nu s-ar putea evita

întârzierile și coliziunile. În cazul calculatoarelor, rolul unui sistem de operare este de a coordona traficul și accesul la resurse, întocmai precum turnul de control al unui aeroport.

1.1.1 Funcțiile unui sistem de operare

O primă funcție a unui sistem de operare constă în **administrarea resurselor hardware ale sistemului**. În cazul unui calculator, este vorba mai ales despre accesul la procesor, memorie, hard-disk, comunicarea în rețea, precum și cu alte dispozitivele de intrare/ieșire. Aceste resurse pot fi solicitate de aplicații multiple și de utilizatori multipli.

Sistemul de operare este la rândul său o aplicație, care necesită resurse (și, implicit, accesul la acestea) pentru a funcționa. Prin urmare, acesta trebuie să își asigure accesul privilegiat la resurse. Un SO trebuie să mențină **controlul asupra calculatorului** și să facă posibilă utilizarea sa predictibilă și cât mai eficientă.

Sistemul de operare oferă o **abstractizare a resurselor hardware**. Practic SO este o extensie a sistemului fizic, oferind o inferfață simplificată folosită pentru medierea accesului utilizatorilor și aplicațiilor la resursele hardware.

Sistemul de operare este responsabil cu oferirea unui set limitat de informații și funcții de lucru cu hardware-ul, în loc de descrierea exactă a ceea ce se întâmplă în hardware. Practic, SO oferă dispozitivelor și aplicațiilor (și, în ultimă instanță, utilizatorilor) o **transparență față de modul de funcționare a hardware-ului**.

În ceea ce privește comunicarea dintre hardware și aplicații, sistemul de operare trebuie să creeze o identitate calculatorului, care să rămână relativ stabilă în dialogul acestuia cu aplicații diverse, chiar dacă au loc upgrade-uri de hardware. Sistemul de operare are deci rolul de **asigurare a securității și integrității sistemului de calcul**. Sistemul de operare va controla accesul utilizatorilor și proceselor la resursele hardware ale sistemului și va preveni execuția de instrucțiuni invalide, malițioase sau privilegiate.

Sistemul de operare este și cel care permite unei aplicații să ruleze pe sisteme fizice diferite, deoarece uniformizează răspunsurile date de calculator și intermediază accesul la resurse. Sistemele de operare trebuie prin urmare să poată interacționa cu arhitecturi hardware și cu periferice foarte diverse, el fiind cel care facilitează accesul aplicațiilor la resursele hardware. Această funcție se numește **portabilitate** și se referă la oferirea unei interfețe unice pentru utilizator, indiferent de diferențele hardware dintre arhitecturi.

Un SO folosește **drivere** pentru comunicarea cu dispozitivele periferice.

> **Driverele** sunt programe care traduc comenzile sistemului de operare în comenzi inteligibile echipamentelor, precum și ieșirea acestora în mesaje accesibile sistemului.

Rolul SO a fost extins prin punerea la dispoziția aplicațiilor a unei colecții de biblioteci (funcții) pe care programele le pot folosi pentru a rezolva probleme specifice legate de interacțiunea cu componentele calculatorului.

Sistemul de operare va include utilitare de diagnosticare și monitorizare a funcționării diferitelor componente atât hardware cât și software. Un sistem de operare poate,

desigur, să eșueze în realizarea funcțiilor sale. De exemplu, un tip foarte frecvent de eroare constă în blocarea funcționării calculatorului. Un astfel de "crash" poate fi cauzat de unele imperfecțiuni ale sistemului, dar poate fi favorizat și de funcționarea necorespunzătoare a anumitor aplicații. Un al doilea tip frecvent de eroare se referă la problemele de securitate, care pot permite intruziunea unui utilizator sau a unei aplicații malițioase. De asemenea, este posibil ca un sistem de operare să nu reușească să interacționeze corect cu anumite periferice sau echipamente. În principal, funcționarea corectă a unui sistem de operare poate fi păstrată prin realizarea metodică a unui număr limitat de operații:

- repararea problemelor (bug-urilor) din sistemul de operare însuși;
- repararea bug-urilor din aplicații;
- eliminarea virușilor, viermilor, a spyware etc;
- instalarea de versiuni actualizate ale sistemului de operare și aplicațiilor acestuia.

1.1.2 Tipuri de sisteme de operare

Există diverse criterii de clasificare a sistemelor de operare. Clasificarea de mai jos este realizată ținând cont de sistemele fizice pe care rulează și domeniul de utilizare.

Sistemele de operare de pe smart carduri sunt cele mai simple, servind deseori funcții reduse (precum în cazul cărților de debit) și administrând resurse foarte limitate. Totuși, ele oferă un management de bază al acestor resurse, inclusiv a spațiului de stocare, putând oferi și servicii de securitate/criptare a datelor ce le dețin.

Sistemele de operare embedded sunt incluse în dispozitive precum televizoarele sau telefoanele mobile, având constrângeri specifice funcționalității lor. Sistemele de operare pentru dispozitivele embedded variază de la sisteme dedicate unui singur scop, care nu oferă o interfață de interacțiune utilizatorului, până la sisteme multitasking (precum în cazul sistemelor de operare pentru PDA) care oferă multe dintre fucționalitățile sistemelor de operare pentru calculatoarele personale. **Sistemele de operare în timp real** (RTOS – Real Time Operating Systems) sunt folosite mai ales în echipamente științifice, industriale și de precizie. Scopul principal al acestora este să optimizeze utilizarea resurselor echipamentelor și să asigure predictibilitatea funcționării lor, dat fiind că eșecurile în coordonarea subcomponentelor pot avea consecințe dezastruoase.

Timpul este o variabilă esențială, deoarece anumite sarcini trebuie realizate într-un interval precis pentru a se putea coordona cu alte sarcini. Astfel de sisteme de operare nu sunt în general menite să permită comunicarea cu utilizatorul, dat fiind că nu este necesar să fie modificate, ele fiind concepute strict pentru executarea anumitor procese tehnologice.

RTOS pot fi împărțite în sisteme în timp real hard, prevalente în procesele industriale, în care constrângerile temporale sunt întotdeauna prioritare, și sisteme în timp real soft, precum sistemele dedicate multimedia, în care discordanțe temporale minore sunt acceptabile sau insesizabile. Multe din sistemele de operare moderne oferă funcționalități specifice sistemelor de operare în timp real soft.

Există numeroase sisteme de operare de timp real. Exemple sunt RTLinux[1], VxWorks[2], TRON[3] și QNX[4]. Majoritatea sistemelor de operare de timp real sunt folosite în cadrul dispozitivelor embedded. Nu există o linie de demarcație clară între sisteme de operare embedded și sisteme de operare în timp real. Sistemele de operare moderne sunt **sisteme multi-tasking** (permit execuția simultană a mai multor sarcini) și **multi-user** (mai mulți utilizatori pot folosi în același timp resursele sistemului). Sistemele multi-user urmăresc în principal să optimizeze funcționarea calculatorului prin echilibrarea cerințelor utilizatorilor și izolarea problemelor acestora. Există sisteme de operare specializate pentru situațiile de conectare a calculatoarelor în rețea, precum și sisteme de operare specializate pentru aplicațiile care rulează pe un server, deși aceste granițe tind să se estompeze.

În această categorie sunt incluse sistemele de operare precum cele din familiile Microsoft Windows, Unix și Apple Mac OS X, ce au ca scop optimizarea folosirii resurselor atunci când un utilizator rulează concomitent mai multe aplicații sau când mai mulți utilizatori interacționează cu un calculator în același timp. Aceste sisteme de operare sunt cele mai cunoscute de utilizatorii de PC-uri.

O practică uzuală pentru obținerea unei puteri sporite de procesare o reprezintă instalarea de procesoare multiple într-un singur sistem. Pentru managementul eficient al acestora și pentru a se putea folosi întreaga putere de procesare disponibilă, există **sisteme de operare pentru arhitecturi multiprocesor** special concepute pentru procesarea paralelă (simultană). Acestea pot fi variante modificate ale sistemelor de operare pentru servere, optimizate pentru gestiunea paralelă a operațiilor pe care le execută.

O dată cu proliferarea sistemelor de operare multi-core, majoritatea sistemelor de operare contemporane pot fi folosite atât pe arhitecturi monoprocesor, cât și pe cele multiprocesor. Sisteme de operare precum Windows Vista, Windows 7, distribuțiile Linux bazate pe kernel 2.6 și Mac OS X oferă suport pentru sisteme multiprocesor.

1.1.3 Funcționarea sistemelor de operare

Un sistem de operare este responsabil de realizarea mai multor sarcini. Responsabilitățile principale sunt:

- administrarea procesorului;
- administrarea memoriei;
- administrarea echipamentelor și a perifericelor;
- administrarea sistemelor de stocare a datelor;
- interfața cu aplicațiile și interfața cu utilizatorii.

Rolul principal al sistemului de operare constă în impunerea de reguli pentru folosirea coerentă a resurselor (denumită și multiplexare). Aceasta se realizează fie prin

[1] http://www.rtlinuxfree.com/
[2] http://www.windriver.com/
[3] http://www.tron.org/index-e.html
[4] http://www.qnx.com/

alternarea în timp a accesului la resurse (multiplexare în timp), de exemplu în cazul administrării procesorului, fie prin delimitarea și alocarea unor segmente din resurse aplicațiilor ce le vor utiliza simultan (multiplexare în spațiu), de exmplu în cazul memoriei.

Responsabilitățile de mai sus sunt îndeplinite de nucleul sistemului de operare. Pe lângă acestea, sistemele de operare conțin componente și utilitare care permit realizarea unor sarcini precum:

- inițializarea și oprirea proceselor, inclusiv comunicarea între procese;
- organizarea aplicațiilor disponibile pe calculator;
- organizarea fișierelor în directoare;
- vizualizarea și editarea fișierelor, precum și redenumirea, copierea sau ștergerea fișierelor.

Administrarea procesorului

Administrarea procesorului vizează optimizarea accesului diferitelor unități software la resursele procesorului; practic, acest lucru se traduce în alocarea de cuante de timp pentru rularea unei aplicații pe un procesor.

Este important de observat că, deși sistemele de operare multi-tasking permit funcționarea simultană a mai multor aplicații, aceasta este de fapt simulată prin comutarea foarte rapidă de la o aplicație la alta a unui procesor care oferă doar facilități single-tasking. Execuția simultană în adevăratul sens al cuvântului are loc numai pe arhitecturi multiprocesor, unde fiecare procesor la un moment dat executa un proces.

Prin urmare, este esențial ca sistemul de operare să fie capabil de a răspunde la solicitările diferitelor aplicații. De asemenea, sistemul de operare trebuie să minimizeze proporția de timp în care procesorul este utilizat pentru execuția sarcinilor de întreținere, de comunicare internă etc, în favoarea timpului alocat cerințelor directe ale utilizatorului.

Elementele de bază cu care lucrează un sistem de operare (mai exact, elementele care folosesc procesorul) nu sunt aplicațiile, ci procesele și firele de execuție (threads).

Un proces (vezi capitolul 5) este un program care execută o acțiune și este controlat de utilizator, de o aplicație sau de sistemul de operare. O aplicație poate fi ea însăși un proces dar poate fi și inițiatoarea altor procese pentru a comunica cu alte calculatoare sau cu alte aplicații. Sistemele de operare inițiază numeroase procese pentru a administra resursele calculatorului, în majoritatea cazurilor, utilizatorii nu sunt conștienți de acest lucru.

Administrarea memoriei

Administrarea memoriei urmărește să asigure faptul că fiecare proces are suficientă memorie, fără să acceseze spațiul de memorie al altor procese. De asemenea, procesele trebuie să folosească adecvat diferitele tipuri de memorie existentă – ținând cont de accesibilitatea diferențiată a acestora.

Administrarea perifericelor

Administrarea perifericelor și a echipamentelor este realizată prin intermediul driverelor. Acestea sunt programe relativ autonome de sistemul de operare, fiind deseori furnizate de producătorul echipamentelor și instalate separat. Ele permit sistemului de operare să comunice cu echipamente noi fără să fie modificat. Comunicarea cu mediul este intermediată de spații tampon (buffere), în care datele de intrare sunt acumulate urmând să fie transmise procesorului în ritmul accesibil acestuia, evitând congestia atunci când apare o aglomerare datorită multitudinii de aplicații.

Interfața cu utilizatorul și aplicațiile

Graphical User Interface

Funcția de mediere a sistemului de operare are două subcomponente:

- interfața cu aplicațiile (API – *Application Programming Interface*) permite aplicațiilor să acceseze facil resursele hardware ale sistemului fără a controla toate detaliile tehnice ale aplicațiilor;
- interfața cu utilizatorul permite acestuia din urmă să comunice cu calculatorul, fie prin comenzi text (TUI – *Text User Interface*) sau printr-o interfață grafică (GUI – *Graphical User Interface*).

1.2 Scurt istoric al sistemelor de operare

Sistemele de operare pentru PC-urile actuale sunt nu numai rezultatul unei evoluții graduale, ci și al unei schimbări conceptuale. Dacă în urmă cu 30 de ani sistemele de operare dispuneau de funcționalități limitate, în zilele noastre sistemul de operare este acompaniat de interfața grafică și de aplicații diverse cu scopul de a fi cât mai accesibil utilizatorului individual neprofesionist. Sistemele de timp real și cele dedicate (embedded) sunt mai apropiate de formele inițiale, dezvoltate în anii '60-'70.

Din punct de vedere a evoluției sistemelor de operare pot fi distinse patru mari perioade, denumite și generații. Aceste generații se află în legătură strânsă cu dezvoltarea sistemelor fizice și a limbajelor de programare.

1.2.1 Prima generație

În prima perioadă, care s-a desfășurat în mare între anii 1945 și 1955, se dezvoltă prima generație de calculatoare, având la bază tehnologia tuburilor vidate. O inovație majoră este introducerea cartelelor perforate, ca modalitate de stocare a informației. În această etapă nu există niciun sistem de operare și nici măcar limbaje de programare; utilizatorul final este responsabil pentru interacțiunea directă cu hardware-ul.

1.2.2 A doua generație

Între anii 1955 și 1965 introducerea tranzistorului marchează trecerea la a doua generație de calculatoare, dominate de sistemele mainframe. Principalele limbaje utilizate sunt FORTRAN și limbajul de ansamblare.

Unitatea operațională principală de structurare a informației în interacțiunea cu calculatorul era **job-ul**, care consta într-unul sau mai multe programe scrise întâi pe hârtie și apoi introduse pe cartele.

O proporție substanțială din timpul de utilizare al mainframe-ului se pierde cu introducerea fizică a joburilor în calculator (prin realizarea și mutarea cartelelor). Pentru a reduce acest incovenient se dezvoltă tehnica procesării agregate a job-urilor (în batch-uri): mai multe job-uri sunt grupate într-un singur șir de cartele separate de delimitatori. Job-urile sunt ulterior imprimate pe bandă magnetică folosind un calculator relativ ieftin, apoi benzile sunt transmise mainframe-ului, urmând ca outputul să fie tipărit din nou prin intermediul calculatorului mai ieftin. Primul sistem de operare major bazat pe principiul "single stream batch processing" a fost dezvoltat de General Motors Research Laboratories pentru mainframe-ul IBM 701 în 1956.

1.2.3 A treia generație

A treia generație de calculatoare, bazată pe circuitele integrate, apare în perioada 1965-1980. Unul dintre cele mai importante inovații al acestei perioada este principiul utilizării în comun a timpului de procesor (timesharing). Conform acestui principiu, mai mulți utilizatori pot folosi simultan terminale conectate la același calculator.

În timesharing fiecare utilizator primește alternativ o fracțiune de timp, perioadă în care procesorul execută instrucțiunile acestuia. Astfel se valorifică eficient timpii morți datorați vitezei scăzute de acțiune a oamenilor față de cea a procesorului, reducând substanțial durata de elaborare și corectare a unui program.

Primul sistem bazat pe timesharing a fost CTSS – Compatible Time-Sharing System, realizat de MIT în 1961. Din sistemul de operare Multics, dezvoltat ulterior tot de MIT, au derivat primele variante de Unix, inițial sub conducerea lui Ken Thompson. Dennis Ritchie a elaborat limbajul C special pentru redactarea Unix, evitând astfel dependența de caracteristicile hardware, inerentă în folosirea limajului de ansamblare.

Unix a fost primul sistem de operare scris într-un limbaj de nivel superior, devenind astfel portabil de la un tip de calculator la altul. Datorită nevoii de standardizare a multiplelor variante de sisteme Unix care au apărut apoi independent pe baza codului sursă, IEEE a elaborat standardul POSIX. Ulterior, Linus Torvalds a scris Linux.

În 1968 a fost creată firma Intel, specializată în microprocesoare. CP/M (Control Program for Microcomputers), elaborat în 1976 de Gary Kildall, este primul sistem de operare pentru calculatoare bazate pe procesoare Intel.

1.2.4 A patra generație

După 1980, până în prezent, se poate vorbi despre a patra generație de calculatoare, reprezentate de apariția și expansiunea extraordinară a calculatoarelor personale. Producția acestora are la bază tehnologia microprocesoarelor.

Bill Gates a cumpărat sistemul de operare DOS (Disk Operating System) și l-a adaptat cerințelor IBM creând MS-DOS. Sistemul a câștigat o poziție dominantă pe piață datorită deciziei lui Gates de a-l direcționa către companiile producătoare de hardware, în special IBM.

În 1981 au apărut primele calculatoare personale IBM, costând 2880 USD, incluzând 64 KB de memorie și un floppy disk. Piața a crescut rapid, în scurt timp apărând și alte modele care foloseau aceeași arhitectură, dat fiind că IBM a decis să o facă publică. Vasta majoritate a acestor calculatoare rulau sub MS-DOS.

Doug Engelbart a realizat interfața grafică pentru utilizatori – GUI, care a fost apoi încoporată în echipamentele construite de Xerox PARC. Steve Jobs, co-inventatorul calculatorului Apple, a transformat GUI într-un succes comercial remarcabil prin lansarea Apple Macintosh în 1984 (dotat cu mouse) și direcționarea acestuia către utilizatorii cu cunoștințe sau aptitudini tehnice minimale.

Succesul lui Macintosh a inspirat Microsoft în elaborarea Windows, conceput inițial ca o interfață grafică peste MS-DOS. În 1995 a apărut prima versiune autonomă de Windows, urmând apoi Windows 98. Windows NT, o versiune complet nouă de Windows, s-a impus treptat pe piața utilizatorilor corporativi, fiind orientat spre servere și oferind posibilități avansate de administrare. A fost urmat de Windows 2000 și Windows 2003 Server. În 2001 arhitectura NT a migrat către utilizatorii individuali prin lansarea lui Windows XP. Prin Windows Vista, și actualmente Windows 7, se dorește păstrarea acestor orientări simultane atât spre partea de server cât și spre partea de utilizator.

După anii '90, sistemele Unix (în special Linux) încep să câștige teren, preponderent pe piața sistemelor server, iar apoi piața sistemelor desktop. La sfârșitul anilor '80, la MIT a apărut o primă interfață grafică pentru sistemele Unix – X Windows, urmată de versiuni mai complexe.

1.3 Sisteme de operare moderne

Deși există mai multe criterii de clasificare, doar anumite tipuri de sisteme de operare vor avea relevanță pentru utilizator obișnuit al unui calculator, fie el și unul cu profil tehnic. Un criteriu pragmatic, care ține cont de răspândirea sistemelor de operare actuale, conduce la clasificarea sistemelor de operare în:

- **sisteme de operare desktop**, destinate sistemelor de calcul obișnuite (desktop) și laptop-urilor;
- **sisteme de operare server**, destinate sistemelor de calcul folosite în general în Internet pentru a oferi servicii;
- **sisteme de operare pentru dispozitive mobile**, destinate telefoanelor mobile, smartphone-urilor sau dispozitivelor de tip PDA sau Palm.

1.3.1 Sisteme de operare desktop și server

Piața sistemelor de operare desktop este dominată în ultimele decenii de sistemele de operare din familia Windows[1]. În prezent, familia Windows deține o cotă de circa 93% din piața sistemelor desktop. Mac OS X deține circa 4.5%-5%, iar Linux în jur de 1%. Toate aceste sisteme de operare rulează în principal pe arhitecturi PC (x86 sau x86_64). Deși cu o cotă pe piață foarte mică raportat cu familia Windows, Mac OS X și Linux au câștigat procente importante pe piață.

Ținând cont că un utilizator poate opta între mai multe sisteme de operare, apar întrebări firești legate de asemănările și deosebirile dintre ele, care sunt punctele tari și punctele slabe ale fiecăruia și, în ultimă instanță, care dintre ele este cel mai bun. În realitate însă, a compara aceste cele trei clase de sisteme este un lucru foarte dificil, date fiind diferențele fundamentale de concept și realizare, iar mai mult, a trage concluzia că unul dintre sisteme este implicit mai bun sau mai slab decat celălalt este imposibil.

Toate cele trei familii de sisteme de operare oferă performanțe avansate, un mediu stabil de operare și au un domeniu foarte bogat de aplicații disponibile și concepute special pentru acestea. Dar pentru că filozofiile din spatele manierei în care sistemele au fost proiectate și au evoluat sunt diferite, concluzia finală depinde în mare măsură de scopul în care sistemul va fi folosit, gradul de pregătire al utilizatorilor, configurația hardware și mulți alți factori subiectivi.

Sistemele de operare din familia Windows și Unix/Linux oferă variante atât pentru sisteme desktop cât și pentru sisteme server. Astfel, Windows XP, Windows Vista, Windows 7 sunt versiuni desktop ale sistemului de operare de la Microsoft, în vreme ce Windows Server 2003, Windows Server 2008 sunt versiuni dedicate pentru sisteme server. În lunea Unix/Linux diferența este mai puțin clară, un sistem putând fi configurat pentru a funcționa atât ca server cât și ca desktop. Anumite distribuții Linux sunt, însă dedicate pentru sisteme desktop (Ubuntu, Fedora, openSUSE, Mint, PCLinuxOS) sau pentru sisteme server (Ubuntu Server, RedHat Enterprise Linux, SUSE Linux Enterprise). Alte sisteme de operare din familia Unix (OpenSolaris, FreeBSD, NetBSD, OpenBSD) pot fi folosite și ca sistem desktop și ca sistem server. Apple oferă Mac OS X ca sistem desktop și Mac OS X Server ca sistem server.

Windows vs. Linux

Un subiect care stârnește mult interes și dezbateri este diferența dintre Windows și Linux. De multe ori această diferență este una de ordin semantic.

Microsoft Windows desemnează un produs bine delimitat, având asociate niște specificații, pe baza cărora există un preț pe piață.

De cealaltă parte, Linux nu reprezintă un produs bine definit, termenul având mai multe înțelesuri. Tehnic vorbind, Linux este numele nucleului (kernel-ului) sistemului de operare. Ca fapt divers, Linux este o marcă înregistrată a lui Linus Torvalds, inițiatorului proiectului nucleul Linux. În prezent, proiectul este încă sub coordonarea lui Linus Torvalds. Nucleul nu depinde de restul sistemului, așa cum nici restul pachetelor care compun sistemul nu depind în mod necesar de kernel, cu unele excepții notabile.

[1] http://en.wikipedia.org/wiki/Usage_share_of_desktop_operating_systems

Cu toate acestea, majoritatea sistemelor au ajuns să fie construite pe serie comună de pachete, bazată pe proiectul GNU[1], întreg sistemul fiind pus laolaltă în ceea ce se cheamă o distribuție: o colecție de pachete, împreună cu nucleul Linux, de întreținerea și actualizarea careia se ocupă o anumită organizație sau comunitate.

Astfel, printr-un abuz de limbaj unanim acceptat, aceste clase de sisteme de operare sunt denumite distribuții Linux (sau GNU/Linux), dată fiind legătura puternică cu nucleul și pachetele de bază construite peste acesta. Întrucât Linux este numele nucleului sistemului de operare, recomandăm folosirea sintagmei *nucleul Linux* sau *kernel-ul Linux* în loc de *nucleul de Linux* sau *kernel-ul de Linux*.

1.3.2 Familia Windows

Microsoft a lansat la început Windows ca pe o colecție de aplicații MS-DOS, rulate într-un mediu grafic interactiv, care să ușureze munca utilizatorului de a lucra cu sistemul. Primele versiuni folosite pe scară largă au fost Windows 3.0 și Windows 3.1.

Un punct important l-a constituit lansarea Windows 95 în 1995. Cu o interfața intuitivă și prietenoasă, suita bogată de aplicații preinstalate, suportul tehnic dedicat, Microsoft a promovat Windows 95 și edițiile ulterioare ca fiind alegerea ideală pentru sistemul desktop de uz personal sau pentru birou.

Acest atu a fost păstrat și la edițiile ulterioare de Windows, unde s-au făcut progrese semnificative în direcția performanțelor: sistemul s-a desprins de MS-DOS, capătând un kernel modern, cu suport de multitasking și mai mulți utilizatori, interfața grafică a devenit mai bogată și mai atrăgătoare și, nu în ultimul rând, necesitatea de cunoștinte tehnice din partea utilizatorilor s-a diminuat de la ediție la ediție.

Edițiile actuale de Windows sunt construite pe tehnologia NT. Din familia NT fac parte Windows NT, Windows 2000, Windows XP, Windows Vista, Windows 7 și versiunile server lansate de Microsoft. Tehnologia NT a însemnat rescrierea nucleului sistemului de operare și obținerea unui sistem de operare modern cu funcționalități de integritate și securitate bine definite.

În momentul scrierii acestei cărți, ultima versiune desktop este Windows Vista, iar ultima versiune server este Windows Server 2008. Pe data de 22 octombrie 2009 este planificată lansarea Windows 7.

Cu o cotă impresionantă pe piața sistemelor de operare, Windows are rolul unui sistem de operare universal. Cea mai mare parte a aplicațiilor existente rulează pe sistemele Windows. Cota mare de piață a făcut Windows ținta a numeroase atacuri de securitate, rezultând în creșterea eforturilor Microsoft pentru asigurarea integrității sistemului la ultimele versiuni de Windows.

1.3.3 Mac OS X

Mac OS X reprezintă o familie de sisteme de operare concepute special pentru calculatoarele Macintosh ("Mac"-uri) produse de firma Apple. Introduse inițial în 1984,

[1] http://www.gnu.org/

sistemele de operare pentru Macintosh au cunoscut o evoluție spectaculoasă până în prezent, menținându-se drept concurent pentru Microsoft Windows atât prin performanțele sistemului propriu-zis cât și prin versatilitatea și utilitatea aplicațiilor pe care le dețin.

Calculatoarele Macintosh au implementat, de la lansare, o arhitectură hardware de tip Power PC, migrând mai apoi către o arhitectură Intel, moment în care sistemele de operare pentru Mac-uri au început să reprezinte o concurență pentru Windows.

Sistemul de operare a fost radical schimbat la lansarea versiunii 10.0, prin rescrierea nucleului. Această schimbare este reflectată și în modificarea denumirii sistemului de operare din Mac OS în Mac OS X. Adăugarea literei X subliniază trecerea de la tehnologia Mac OS nanokernel, la tehnologie derivată din Unix.

În prezent, varianta curentă de Mac OS X este 10.6, lansată la data de 28 august 2009.

Prin sistemul de operare și prin aplicațiile care pot rula pe acesta, Mac OS X excelează la capitolul interfață grafică și în domeniul prelucrării media, aducând în același timp inovații orientate spre nevoile utilizatorului individual.

1.3.4 Linux

Una dintre principalele confuzii pentru un utilizator proaspăt intrat în lumea sistemelor de operare este cea provocată de denumirile Linux și Unix. În general, Unix reprezintă o familie de sisteme de operare. Un sistem de operare este Unix (adică face parte din familia Unix) dacă respectă standardul *Single Unix Specification*[1]. În afară de Linux (sau mai bine spus GNU/Linux), alte Unix-uri (adică sisteme compatibile cu Single Unix Specification) sunt Solaris, sistemele de familia BSD (FreeBSD, OpenBSD, NetBSD) și Mac OS X.

Sistemul de operare Unix a apărut la începutul anilor '70, în laboratoarele Bell ale AT&T. Folosirea limbajului C pentru scrierea sistemului de operare a condus la portarea rapidă a acestuia și răspândirea sa în mediul acadamic. Versiunile System V (de la AT&T) și BSD 4.3 au marcat o serie de avansuri importante în anii '80.

Linux este de fapt o clonă de Unix. O clonă Unix este un sistem de operare care respectă Single Unix Specification, dar care nu conține componente din sursele inițialte Unix. BSD-urile sau Solaris sunt direct derivate din codul Unix și pot fi denumite Unix-uri veritabile.

Linux a fost scris de la zero de Linus Torvalds în 1991. Treptat, în jurul său s-a consolidat o comunitate entuziastă de dezvoltatori, Linux ajungând în acest moment unul dintre cele mai mari proiecte open-source. Dincolo de funcționalitățile oferite, popularitatea Linux în ultimii 15 ani s-a datorat apariției acestuia într-un context favorabil:

- dezvoltatorii proiectului GNU nu reușiseră crearea unui kernel;
- cei care urmăreau dezvoltarea viitoarelor sisteme BSD sub o licență liberă au pierdut câțiva ani în procese cu cei de la AT&T legate de licențierea codului.

Linus Torvalds a spus că dacă 386BSD ar fi fost disponibil la acea vreme, probabil că nu ar fi creat Linux.

[1] http://en.wikipedia.org/wiki/Single_UNIX_Specification

După apariția nucleului Linux, o multitudine de proiecte software având la bază acest kernel au fost înființate, cu numărul și importanța acestora crescând de la an la an. Spre deosebire de cazul Windows-ului, software-ul pe Linux nu a fost scris neapărat având în minte ușurința de utilizare și accesibilitatea, deși aceasta este tendința din prezent.

Un nucleu este doar un mediator între aplicații și resursele hardware – este inutil de unul singur. Pentru a obține un sistem de operare complet, diverse comunități sau companii au combinat utilitarele GNU și nucleul Linux. Au apărut astfel distribuțiile GNU/Linux (Ubuntu, Fedora, Gentoo, Debian, Slackware, OpenSuSE, RedHat etc.), care oferă utilizatorului funcționalități complete pentru operarea sistemului. Din punct de vedere strict tehnic, expresia corectă este *instalarea unei distribuții Linux (GNU/Linux)*, nu *instalarea Linux*. Ținând cont de prevalența formei *distribuție Linux*, nu vom insista asupra acestui lucru pe parcursul acesi carți pe justețea exprimării (vezi secțiunea 2.1).

Fiind vorba îndeosebi de mediul academic, majoritatea utilizatorilor aveau cunoștințe avansate de IT și apreciau la un sistem stabilitatea, flexibilitatea și performanța, mai degrabă decât o interfață simplă și intuitivă. Datorită acestor puternice avantaje, Linux (ca majoritatea sistemelor Unix, de altfel) a avut un succes exploziv pe servere, și a câștigat teren într-o măsură mult mai mică pe desktop, unde supremația este încă deținută de către sistemele de operare de la Microsoft. În momentul de față, o bună parte din infrastructura Internet-ului se bazează pe sisteme rulând Linux.

1.3.5 Alte Unix-uri

Având la bază un kernel Mach[1] și facilități BSD, **Mac OS X** poate fi considerat un sistem Unix-like. De la Mac OS X 10.5 Leopard, sistemul de operare de la Apple respectă *Single Unix Specification* și este un Unix veritabil[2].

Solaris[3] este sistemul de operare dezvoltat de Sun Microsystems. Inițial dezvoltat în format proprietar (*closed-source*), majoritatea componentelor sunt acum open-source. **OpenSolaris**[4] este o distribuție open-source a sistemului de operare de la Sun, utilizabil și în forma unui LiveCD.

Distribuțiile **BSD** (*Berkeley Software Distribution*) sunt distribuțiile derivate din codul 4.4BSD-Lite, ultima versiune de Unix de la Berkeley. Aceste distribuții folosesc o licență liberă (licență BSD). Licența fiind permisivă, o parte din componentele software sunt integrate în Windows, Mac OS X și Solaris. Distribuțiile BSD sunt asociate unei distribuții GNU/Linux: nu conțin doar nucleul, ci și toate aplicațiile necesare folosirii sistemului. Dezvoltarea întregii distribuții se realizează centralizat. Distribuțiile BSD cele mai folosite astăzi sunt:

- **FreeBSD** – cel mai folosit sistem de operare derivat din BSD – axat pe oferirea unui sistem de operare stabil și utilizabil;
- **OpenBSD** – sistem de operare derivat din NetBSD, puternic axat pe securitate;

[1] http://en.wikipedia.org/wiki/Mach_kernel
[2] http://arstechnica.com/apple/news/2007/08/mac-os-x-leopard-receives-unix-03-certification.ars
[3] http://www.sun.com/software/solaris/
[4] http://opensolaris.org/os/

- **NetBSD** – recunoscut pentru portabilitatea sa; este folosit ca punct de plecare pentru portarea altor sisteme de operare pe noi arhitecturi.

1.3.6 Comunități open source

Principala diferență dintre Linux și multe alte sisteme de operare actuale, mai mult sau mai puțin răspândite, este faptul că atât nucleul Linux cât și componentele unui sistem GNU/Linux sunt gratuite. Mai mult decât atât, sunt free/open-source. Acest lucru înseamnă că o comunitate vastă de utilizatori și ingineri software au acces la ele, pentru a le utiliza sau a le aduce noi îmbunătățiri. Acestă practică nu este nouă, ci datează de la începuturile Unix la MIT labs, când echipele inițiale de hackeri lucrau în comun pentru dezvoltarea de aplicații Unix.

Conceptul de licență sub care funcționează Linux, alături de aplicațiile sale, a primit denumirea de *copyleft* care, contrar copyright-ului, oferă oricui dreptul de acces, distribuție și modificare a unei creații, atâta timp cât păstrează aceleași caracteristici ale *copyleft-ului*. La polul opus, majoritatea sistemelor de operare ce nu aparțin familiei Linux și, cu preponderență cele comerciale, se supun în continuare legii copyright-ului și împiedică intervenția utilizatorilor asupra codului sursă (closed source).

Comunitățile open source sunt foarte dinamice, iar, în acest moment, distribuțiile de Linux beneficiază din plin de acest lucru. Până acum câțiva ani, sistemele Linux necesitau cunoștințe avansate de operare, din motive legate de instrumente grafice de administrare limitate sau lipsa driverelor din partea producătorilor de hardware. Mai mult, accesul limitat la Internet priva utilizatorul obișnuit de accesul la documentație (deși vastă, în mare parte online), esențială pentru încurajarea utilizatorilor de a experimenta cu un nou sistem de operare, în contextul în care nu existau campanii de marketing ca în cazul produselor comerciale.

Lucrurile au început totuși să se schimbe într-un mod radical în momentul în care Internet-ul a devenit o resursă din ce în ce mai accesibilă și o serie de companii au remarcat potențialul pe care Linux îl avea și și-au orientat domeniul de activitate către acesta. Interesul crescut a încurajat comunitățile să depună un efort mai mare pentru a face aplicațiile mai accesibile.

Robustețea și performanțele superioare au fost îmbinate cu o interfață bogată și o experiență vizuală comparabilă (dacă nu superioară – vezi proiectele Compiz și Beryl) cu cea de pe Windows. Suportul din partea producătorilor de hardware a crescut la rândul său, cu implicări foarte puternice din partea unor nume ca Novell, Canonical, sau Dell. Astfel, desktop-ul Linux devine o opțiune atrăgătoare, anii următori anunțându-se extrem de promițători pentru ascensiunea mișcării Open Source în mediul desktop.

Există, totuși, reticențe cu privire la încurajarea răspândirii programelor open-source din moment ce o astfel de libertate a tuturor utilizatorilor este dificil de controlat și poate crea probleme de interoperabilitate. Într-adevăr, unificarea pe care o oferă sistemele de operare comerciale, concepute global și, totodată, compact, impune și uniformizarea aplicațiilor scrise pentru a rula pe ele, în timp ce varietatea deja mare și în continuă creștere a sistemelor open-source conduce la stabilirea unei multitudini de direcții de dezvoltare între care se împarte software-ul conceput pentru toate acestea.

Pe de altă parte, comunitatea de utilizatori ai programelor open-source este în continuă

creștere iar dezvoltarea intensă a distribuțiilor Linux conduce la crearea de pachete specifice care facilitează accesul la aplicațiile necesare. În momentul de față, versiunea stabilă a distribuției Debian GNU/Linux oferă peste 25 de mii de pachete[1].

Cea mai mare comunitate de programatori ce dezvoltă aplicații open-source pentru un număr foarte mare de platforme și sistemele de operare este SourceForge[2]. Microsoft a încurajat dezvoltarea unei comunități, numită CodePlex[3], axată pe dezvoltarea aplicațiilor open-source folosind tehnologiile Microsoft, precum .NET Framework, Windows Presentation Foundation, XNA etc. pentru sisteme de operare de la Microsoft, cu preponderență Window XP și Windows Vista.

1.3.7 Sisteme de operare pentru dispozitive mobile

O dată cu dezvoltarea hardware-ului și a Internet-ului, un număr din ce în ce mai mare de dispozitive mobile inteligente și-au făcut apariția în posesia utilizatorilor. Așa numitele *smartphone-uri*[4] posedă sisteme de operare cu facilități avansate de conectare la Internet, navigare pe Web, citire/scriere de e-mail-uri, folosirea unei tastaturi virtuale etc. Exemple de astfel de dispozitive sunt iPhone[5] de la Apple, BlackBerry[6], Nokia E71[7], HTC[8] etc.

Cel mai răspândit sistem de operare folosit pe dispozitive mobile în general, și pe smartphone-uri în particular este **Symbian OS** de la Symbian Ltd. Acesta deține aproape 50% din piața sistemelor de operare pentru smartphone-uri. Sistemul conține kernel-ul și o serie de aplicații utile în lucrul cu telefonul mobil. Symbian OS oferă o interfață de dezvoltare (SDK) în C++ pentru scrierea de noi aplicații. Înființată în 2008, Fundația Symbian are ca scop folosirea unei licențe libere pentru întreg codul platformei Symbian.

Windows Mobile este sistemul de opearare și suita de aplicații pentru dispozitive mobile de la Microsoft. Este proiectat să arate simiar interfeței Windows și este folosit cu preponderență în cadrul dispozitivelor Pocket PC. Cota de piață a Windows Mobile a scăzut constant în ultimii ani până la o valoare circa 9%.

iPhone OS este sistemul de operare dezvoltate de Apple pentru iPhone și iPod. Interfața acestuia folosește gesturi de tip multi-touch – se atinge ecranul pentru acționarea unui eveniment. iPhone OS oferă un număr impresionant de aplicații. Deține, în acest moment, aproape 15% din piața dispozitivelor mobile inteligente.

Android este un sistem de operare care folosește nucleul Linux. Android este dezvoltat de *Open Handset Alliance*, un consorțiu înființat de Google împreună cu mari companii producătoare de hardware și software precum HTC, Intel, Motorala, LG etc. Dezvoltarea acestuia se realizează în cadrul unei licențe libere (cu unele excepții), lucru care permite dezvoltarea relativ facilă de noi aplicații. În aceste momente, Android are o cotă de

[1] http://packages.debian.org/lenny/
[2] http://sourceforge.net
[3] http://www.codeplex.com/
[4] http://en.wikipedia.org/wiki/Smartphone
[5] http://www.apple.com/iphone/
[6] http://www.blackberry.com/
[7] http://europe.nokia.com/find-products/devices/nokia-e71
[8] http://www.htc.com/www/product/dream/overview.html

piață de doar 3%, cauzată și de apariția sa recentă. Dezvoltarea Android se realizează independent de platforma hardware pe care va rula.

Linux este folosit ca bază pentru mai multe sisteme de operare pentru dispozitive mobile. Pe lângă Android, LiMo, Maemo și Openmoko sunt exemple de sisteme de operare bazate pe Linux.

1.4 Studiu de caz

1.4.1 Virtualizare. Rularea unui SO dintr-o mașină virtuală

O **mașină virtuală** este o aplicație ce creează un mediu de execuție pentru rularea unui nou sistem de operare, folosind resursele gestionate de sistemul de operare gazdă, prin intermediul mecanismelor oferite de mașina virtuală.

O caracteristică particulară a mașinii virtuale este capacitatea acesteia de a izola procesele ce rulează sistemul de operare din interiorul său de cele ale sistemului de operare gazdă.

Sistemul de bază, în cadrul căruia (sau peste care) rulează mașina virtuală se cheamă **sistem gazdă (*host*)**. Mașina virtuală se mai numește și **sistem oaspete (*guest*)**.

Aplicația care permite rularea unui sistem de operare virtualizat se numește **hipervizor** (*hypervisor*) sau *virtual machine monitor* (VMM) sau sistem/soluție de virtualizare.

Exemple de soluții de virtualizare sunt VMware, VirtualBox, VirtualPC, Xen, KVM.

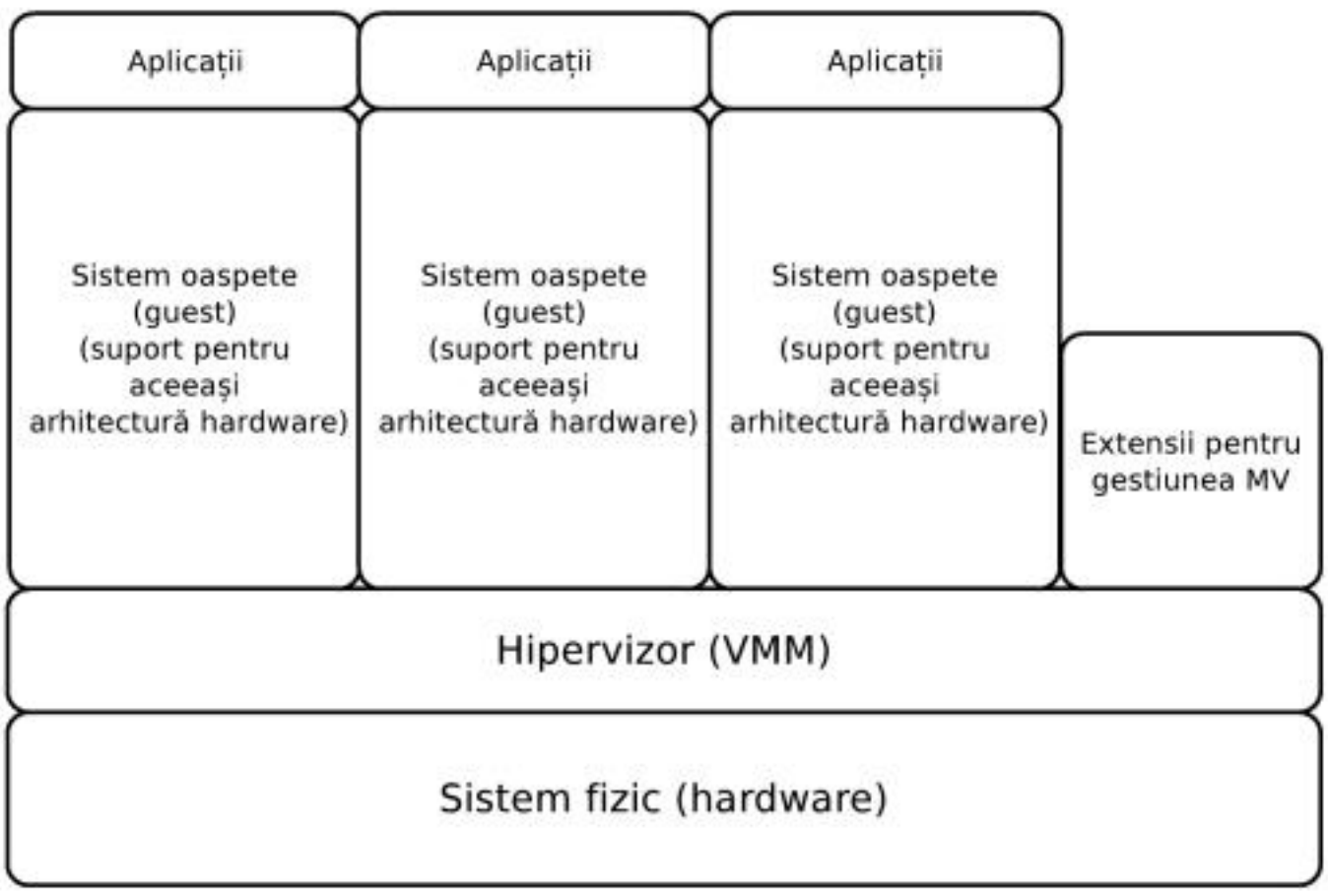


Figura 1.2: Mașină virtuală

În general, sistemul gazdă alocă părți din resursele sistemului fizic pentru mașina virtuală: un procentaj dedicat din memoria RAM, un spațiu fizic de stocare de pe hard-disk-ul calculatorului gazdă (alocat în prealabil sau în timp real) pe care mașina virtuală îl va folosi ca pe un hard-disk propriu, accesul la diferite dispozitive periferice. Accesul celui de-al doilea sistem de operare la procesor se face prin intermediul proceselor mașinii virtuale, care vor rula alături de celelalte aplicații și vor împărți aceleași resurse. Noțiunea de virtualizare nu trebuie confundată cu cea de **emulare**[1]. Un emulator traduce fiecare instrucțiune din sistemul oaspete în instrucțiuni specifice sistemului gazdă. Prin folosirea unui emulator este astfel posibilă rularea unui program pentru o arhitectură diferită de cea a sistemului gazdă. Un sistem virtualizat va folosi în proporție cât mai mare sistemul fizic gazdă, din motive de eficiență. Rolul hipervizorului este de a intermedia unele operații privilegiate care nu se pot executa direct peste sistemul fizic gazdă.

Noțiunea de virtualizare este extinsă și la nivelul aplicațiilor. Programele Java rulează în cadrul mașinii virtuale Java[2]. Mașina virtuală Java este un mediu independent de platformă care permite execuția în același context a aplicațiilor. Un alt exemplu este *Common Language Runtime*[3], mașina virtuală folosită de *.NET Framework*[4]. Valgrind (vezi secțiunea 14.8.2), folosit pentru depanarea programelor, este, de asemenea, o mașină virtuală la nivelul aplicațiilor.

De ce virtualizare?

Dezvoltate încă din anii '70, mașinile virtuale au început să fie folosite intens odată cu dezvoltarea sistemelor hardware. Creșterea capacității de calcul (procesoare multi-core, memorie) și a capacității de stocare (hard-disk-uri) a condus la apariția soluțiilor de virtualizare care să folosească eficient resursele hardware puse la dispoziție. Adăugarea de suport hardware pentru arhitecturile x86 și x86_64 (AMD-V[5] și Intel VT[6]) a condus la accelerarea folosirii soluțiilor de virtualizare atât pentru utilizatorul obișnuit cât, mai ales, în mediul business.

Unul din principalele motive pentru folosirea virtualizării, mai ales în mediul comercial, este **consolidarea încărcării sistemelor fizice**. În locul folosirii a patru sisteme fizice, se poate folosi un singur sistem fizic în cadrul căruia să ruleze patru mașini virtuale, cu efect în diminuarea costurilor de achiziție și administrare.

Un al avantaj importat în reprezintă **securizarea aplicațiilor**. Fiecare mașină virtuală este un mediu izolat și sigur (sandbox). O problemă care apare pe o mașină virtuală nu va afecta niciun alt sistem și va putea fi diagnosticată rapid folosind sistemul gazdă. Virtualizarea este un concept important pentru construirea de platforme de calcul sigure.

Soluțiile de virtualizare permit crearea de instanțe de sisteme de operare sau **medii de execuție cu limitări impuse**. Se pot configura procentaje de resurse care să fie folosite de mașina virtualizată. Se asigură astfel un control fin pentru gestiunea resurselor.

---

[1] http://en.wikipedia.org/wiki/Emulator
[2] http://www.java.com/en/download/index.jsp
[3] http://msdn.microsoft.com/en-us/library/ddk909ch(VS.71).aspx
[4] http://msdn.microsoft.com/en-us/netframework/default.aspx
[5] http://www.amd.com/us/products/technologies/virtualization/Pages/virtualization.aspx
[6] http://www.intel.com/technology/virtualization/

Pentru utilizatorul obișnuit, cel mai important avantaj al virtualizării este **posibilitatea rulării mai multor sisteme de operare**. Sisteme de operare diferite, versiune de sisteme operare diferite, pot rula simultan pe același sistem. Pornirea și repornirea unui sistem de operare oaspete se rezumă la interacțiunea cu soluția de virtualizare fără a implica acțiuni asupra sistemului fizic.

Pentru un dezvoltator, o mașină virtuală reprezintă un **mediu ideal pentru dezvoltare și testare**. Orice problemă a aplicației pe care o dezvoltă/testează va afecta doar sistemul de operare oaspete. Dezvoltarea la nivelul nucleului (kernel programming) se realizează, în general, folosind mașini virtuale. Scenarii complexe de monitorizare a performanței sau depanare pot fi implementate pe o mașină virtuală fără pierderea productivității.

Un avantaj al folosirii mașinilor virtuale este **mobilitatea acestora**. O mașină virtuală este, de obicei, văzută ca un set de fișiere. Mutarea acestora pe un alt sistem de calcul este echivalent cu migrarea mașinii virtuale pe acel sistem. Sistemul de operare din cadrul mașinii virtuale va rula în același mod pe un nou sistem. Este posibilă migrarea unei mașini virtuale în timp ce aceasta rulează, tehnică denumită *Live Migration*[1].

Exemple de mașini virtuale

O dată cu proliferarea tehnologiilor de virtualizare, ultimul deceniu a însemnat apariția mai multor companii și soluții. Printre cele mai cunoscute soluții de virtualizare actuale sunt VMware, VirtualBox, VirtualPC, Xen, KVM, OpenVZ. Aceste soluții de virtualizare rulează, în general, pe arhitecturi x86 sau x86_64.

VMware, Inc.[2] este o companie americană care produce mai multe aplicații de virtualizare. Cele mai cunoscute sunt VMware Workstation, VMware Player, VMware Server și VMware ESX. VMware Workstation, Player și Server sunt aplicații care rulează într-un sistem de operare gazdă (Windows sau Linux). VMware Fusion este versiunea pentru Mac OS X. VMware ESX, versiunea enterprise, este un sistem de operare complet peste care se instalează instanțe de mașini virtuale.

VirtualBox[3] este o soluție de virtualizare dezvoltată de Sun Microsystems. Similar cu VMware Workstation, este o aplicație care rulează peste un sistem de operare gazdă (Windows, Linux, Mac OS X, Solaris) și în cadrul cărei pot fi instalate alte sisteme de operare. VirtualBox Open Source Edition este disponibilă sub licență GNU.

Microsoft Virtual PC[4] este un sistem de virtualizare pentru sisteme de operare din familia Windows. Similar VirtualBox și VMware Workstation, Virtual PC oferă suport pentru sisteme oaspete doar din familia Windows. Se pot rula și sisteme de operare Linux, dar cu pași suplimentari de configurare.

Xen[5] este un hipervizor (virtual machine monitor) creat la Universitatea din Cambridge și dezvoltat actualmente de comunitatea Xen ca software liber, sub licență GNU. Xen este similar VMware ESX. Are suport în nucleul sistemului de operare Linux, NetBSD și Solaris. Peste sistemul de bază pot fi instalate alte sisteme de operare oaspete. Xen

[1] http://en.wikipedia.org/wiki/Live_Migration
[2] http://www.vmware.com/
[3] http://www.virtualbox.org/
[4] http://www.microsoft.com/windows/virtual-pc/
[5] http://www.xen.org/

folosește o forma de virtualizare denumită paravirtualizare care permite performanță ridică. Paravirtualizarea necesită însă modificarea sistemului de operare oaspete pentru a putea rula peste Xen. Dacă sistemul de bază oferă suport de virtualizare hardware, atunci nu mai este necesară modificarea sistemului de operare; efectul este posibilitatea rulării unui sistem Windows nemodificat peste Xen.

KVM[1] (*Kernel-based Virtual Machine*) este o infrastructură de virtualizare a nucleului Linux. KVM oferă suport pentru virtualizarea hardware (Intel VT și AMD-V). KVM este inclus în nucleul Linux începând cu versiunea 2.6.20. O parte din utilitarele necesare sunt furnizate de QEMU[2].

OpenVZ[3] este o tehnologie la nivelul sistemului de operare bazată pe Linux. Licențiat GPL, OpenVZ constituie baza pentru Parallels Virtuozzo Containers, o aplicație proprietară furnizată de Parallels, Inc.[4] OpenVZ este mai degrabă un container decât o mașină virtuală. OpenVZ nu virtualizează întreg sistemul de operare. Nucleul sistemului de operare gazdă este folosit de toate mașinile virtuale (denumite și containere sau medii virtuale). Procesele din cadrul fiecărui container sunt izolate, dar se folosește același nucleu. Prezența aceluiași nucleu înseamnă, totuși, overhead redus pentru lucrul cu diversele containere rezultând în viteză și performanță ridicate.

Cuvinte cheie

- programe de bază
- sistem de operare
- nucleu (kernel)
- portabilitate
- API
- biblioteci
- Windows
- Mac OS X
- Linux
- Unix
- open-source
- smartphone
- virtualizare
- hipervizor
- VMware
- VirtualBox
- VirtualPC
- Xen
- KVM
- OpenVZ

Întrebări

1. Care dintre următoarele aplicații poate fi privită ca o componentă a sistemului de operare?
   - ❑ procesor de text
   - ❑ player pentru fișiere video

[1] http://www.linux-kvm.org/page/Main_Page
[2] http://www.nongnu.org/qemu/
[3] http://wiki.openvz.org/Main_Page
[4] http://www.parallels.com/eu/

- ❑ interpretor de comenzi
- ❑ client de email

2. Care dintre următoarele NU reprezintă o resursă partajată în cazul virtualizării?
   - ❑ RAM
   - ❑ hard-disk
   - ❑ procesor
   - ❑ BIOS
3. Care dintre următorii este autorul unui limbaj de programare?
   - ❑ Larry Wall
   - ❑ Linus Torvalds
   - ❑ William Henry Gates III
   - ❑ Andrew S. Tanenbaum
4. Care dintre următoarele NU este o funcție a sistemului de operare?
   - ❑ controlul accesului la memorie
   - ❑ căutarea și eliminarea programelor virus
   - ❑ asigurarea comunicării între procese
   - ❑ organizarea fișierelor în directoare
5. Care din următoarele sisteme de operare NU este folosit pentru dispozitive mobile?
   - ❑ OpenSolaris
   - ❑ iPhone OS
   - ❑ Symbian OS
   - ❑ Android
6. Care din următoarele tehnologii de virtualizare rulează NUMAI pe un sistem Linux?
   - ❑ VirtualBox
   - ❑ VMware
   - ❑ VirtualPC
   - ❑ OpenVZ
7. Ce reprezintă termenul GNU din *distribuție GNU/Linux*?
   - ❑ o antilopă din Africa
   - ❑ o anagramă a cuvântului *gun* (armă)
   - ❑ un proiect care produce o componentă importantă a aplicațiilor ce rulează peste nucleul Linux
   - ❑ compania care se ocupă de dezvoltarea nucleului Linux

8. Care din următoarele sisteme de operare are cea mai mare cotă pe piața sistemelor desktop?
   - ❑ Windows
   - ❑ Mac OS X
   - ❑ Linux
   - ❑ OpenSolaris
9. Care din următoarele sisteme de operare are cea mai mare cotă pe piața dispozitivelor mobile inteligente?
   - ❑ Windows Mobile
   - ❑ iPhone OS
   - ❑ Android
   - ❑ Symbian OS
10. Care din următoarele sisteme de operare este cel mai vechi?
   - ❑ DOS
   - ❑ Unix
   - ❑ Linux
   - ❑ OpenBSD

Capitolul 2

Instalarea Linux. Configurări de bază

Unix is the answer, but only if you phrase the question very carefully.

Ce se învață din acest capitol?

- Ce este Linux
- Distribuții GNU/Linux
- Partiționarea discului; pregătirea pentru instalare
- Instalarea Kubuntu
- Interfața cu utilizatorul; interfața în linia de comandă
- Autentificarea în sistem
- Configurări de bază într-un sistem Linux
- Interoperabilitate Linux/Windows

2.1 Linux. Distribuții Linux

Termenul "Linux" este deseori folosit pentru a reprezenta o întreagă clasă de sisteme de operare și nu un sistem de operare concret. Inițial, termenul "Linux" a fost folosit pentru a referi nucleul Linux[1] (*Linux kernel*), sistemele de operare bazate pe Linux și pe utilitarele din proiectul GNU[2] fiind numite *sisteme de operare GNU/Linux*. Ulterior, termenul a ajuns să fie folosit pentru a referi atât nucleul Linux cât și programele care funcționează cu acesta.

O mare parte din utilitarele folosite în Linux sunt dezvoltate ca proiecte open source (vezi capitolul 14) și, deși sunt dezvoltate într-o manieră colaborativă, sunt produse independent unele de altele. Licențele sub care se afla aceste proiecte permit

[1] http://www.kernel.org/
[2] http://www.gnu.org/

redistribuirea codului sursă, făcând astfel posibilă apariția unor noi proiecte mai mari, care să adune la un loc programe dezvoltate independent și să le facă disponibile la un loc sub forma unei distribuții.

> O **distribuție Linux** reprezintă astfel o colecție de programe, unele din ele modificate special pentru acea distribuție, care sunt combinate cu o versiune de kernel, de multe ori și aceasta modificată pentru acea distribuție, pentru a crea un sistem de operare.

Distribuțiile Linux au căpătat de-a lungul timpului cele mai variate forme și destinații. Unele au ca scop principal utilizarea lor pe servere, altele sunt destinate utilizării pe Ssisteme desktop, iar altele sunt optimizate pentru a ocupa cât mai puține resurse putând astfel fi folosite în sistemele embedded.

2.1.1 Principalele distribuții Linux

În momentul de față există peste 300 de distribuții active (care sunt dezvoltate în continuare). Dintre ele doar puțin peste 10 sunt cunoscute la scară largă, fiind utilizate pentru scopuri generale.

De cele mai multe ori o distribuție nu este dezvoltată de la zero, punctul de plecare constituindu-l un anumit stadiu al unei alte distribuții. Noua distribuție poate moșteni de la cea din care este dezvoltată diverse elemente, cum ar fi modul în care se instalează aplicații noi, locul în care se găsesc diverse fișiere de configurare, felul în care arată aceste fișiere etc.

Popularitatea distribuțiilor poate fi urmărită pe pagina web http://distrowatch.com. Aici este realizat un clasament al distribuțiilor pe baza numărului de accesări zilnice ale informațiilor despre distribuție pe pagina amintită. Acest clasament nu ia în considerare efortul depus pentru menținerea distribuției, dar arată în mare măsură interesul manifestat de utilizatori.

Cele mai accesate 16 distribuții în ultimul an (la data de 19.09.2009) sunt prezentate în tabelul 2.1 (coloana "Accese" prezintă numărul mediu de accese zilnice pe perioada ultimul an; Distribuția *SLS* este descrisă online[1]).

Pe lângă distribuțiile de uz general (pentru servere sau pentru calculatoare personale) există și distribuții dedicate unui scop precis (*one purpuse*) – spre exemplu **Musix**[2], distribuție dedicată producerii de material multimedia, **SystemRescueCD**[3], distribuție Live CD (vezi secțiunea 2.2.3) specializată în repararea unui calculator pe care sistemul de operare nu se mai inițializează din motive ce țin de software, sau **BackTrack**[4], distribuție Live CD care oferă utilitarele necesare atacării unei rețele de calculatoare.

O clasă aparte de distribuții o reprezintă cele care au ca țintă ocuparea a cât mai puțin spațiu. Cea mai cunoscută distribuție ce urmărește acest lucru este **DamnSmallLinux**[5], distribuție care reușește să includă în doar 50MB o distribuție completă cu interfață grafică și cu toate aplicațiile uzuale.

---

[1] http://en.wikipedia.org/wiki/Softlanding_Linux_System
[2] http://www.musix.org.ar/en/
[3] http://www.sysresccd.org/
[4] http://www.remote-exploit.org/backtrack.html
[5] http://www.damnsmalllinux.org/

Tabelul 2.1: Cele mai accesate distribuții pe distrowatch.com (ultimul an)

Poz.	Nume	Anul apariției	Derivată din	Accese
1	Ubuntu	2004	Debian	2221<
2	openSUSE	1994	Slackware	1454=
3	Fedora	2003	Red Hat	1441>
4	Mint	2006	Ubuntu	1401<
5	Debian	1993	–	1074<
6	Mandriva	1998	Red Hat	970=
7	PCLinuxOS	2003	Mandriva (Mandrake)	853<
8	Puppy	2005	–	718>
9	Sabayon	2006	Gentoo	667>
10	CentOS	2003	Red Hat	626<
11	Arch	2002	–	617>
12	MEPIS	2003	Debian	586=
13	Slackware	1993	SLS	561>
14	FreeBSD	1993	386BSD	491>
15	Kubuntu	2005	Ubuntu	427<
16	Damn Small	2003	Knoppix	423<

Pe lângă acestea, comunitățile au dezvoltat distribuții caracteristice unor anumite zone geografice – **Kiwi Linux**[1] este o distribuție adaptată utilizatorilor din România.

Pe Internet sunt disponibile mai multe diagrame care prezintă evoluția distribuțiilor și felul în care au derivat una din alta. O simplă căutare a șirului de caractere "*linux distro timeline*" pe *Google Images*[2] va oferi câteva zeci de variante ale acestei diagrame. O versiune care conține doar principalele distribuții este prezentată în figura 2.1.

2.1.2 Debian. Ubuntu

Una dintre primele distribuții apărute este **Debian**. Din 1993 și până în prezent această distribuție a reprezentat punctul de plecare a peste 40 de distribuții.

Unul dintre cele mai mari avantaje ale distribuției îl reprezintă suportul bun disponibil pe Internet, atât pe pagina web oficială[3] cât și pe alte site-uri. Distribuția oferă posibilitatea de a instala și configura sistemul de operare folosind pachete, fiind disponibile peste 25000 de pachete[4] – există disponibil suport pentru actualizarea facilă și organizată a sistemului de operare. De asemenea sunt disponibile trei variante ale distribuției: *stable*, *testing* și *unstable*, exitând astfel posibilitatea alegerii gradului de stabilitate și noutate dorit.

Cea mai cunoscută distribuție derivată din Debian este **Ubuntu**[5]. În momentul de față Ubuntu este și distribuția cu cea mai mare popularitate dintre toate distribuțiile existente, fiind pe primul loc în clasamentul distrowatch.com de aproape patru ani. Atuul principal

[1] http://kiwilinux.org/
[2] http://images.google.com/
[3] http://www.debian.org/support/
[4] în versiunea Debian 5.0 (*lenny*)
[5] http://www.ubuntu.com

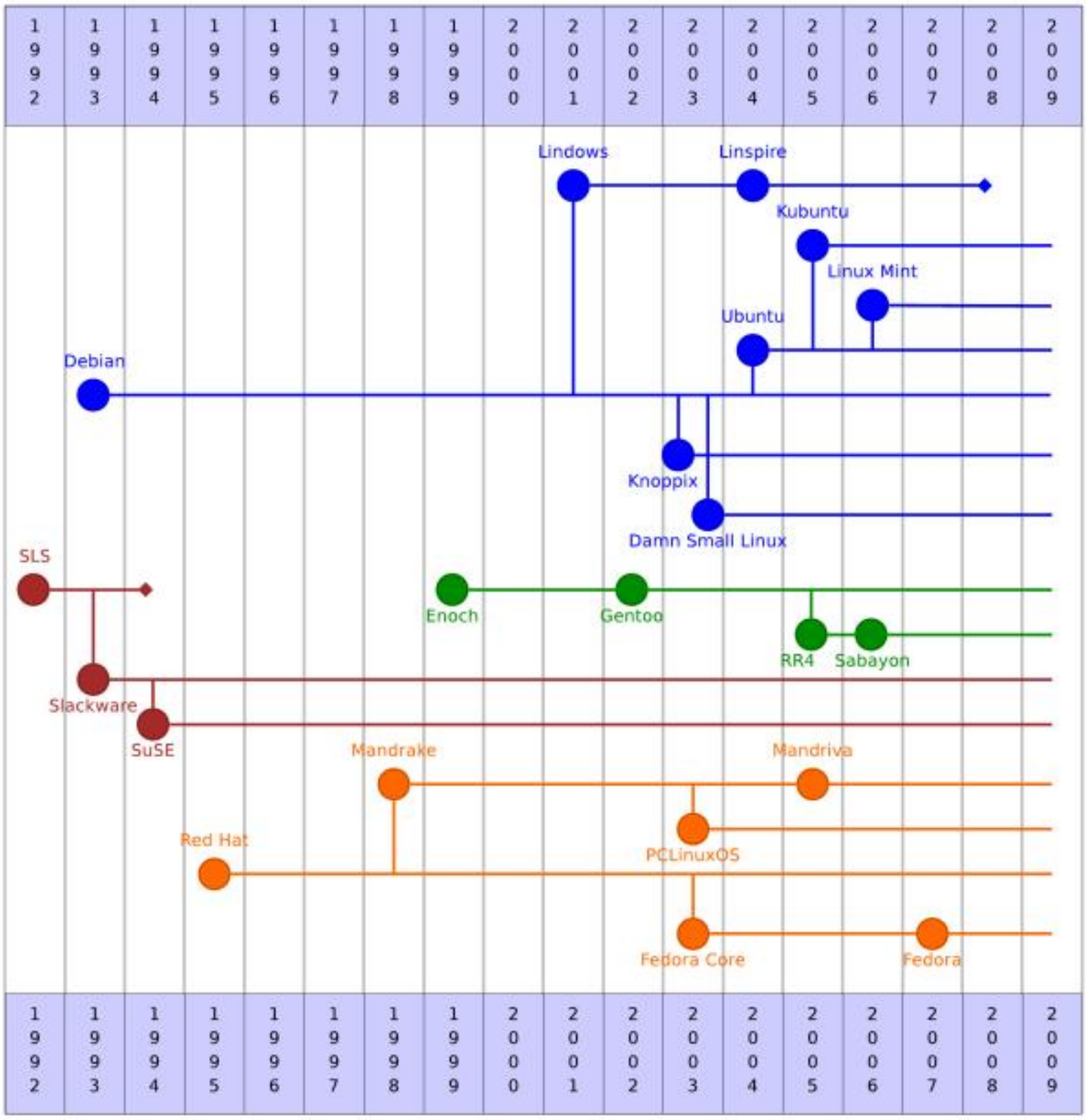


Figura 2.1: Evoluția principalelor distribuții Linux

al acestei distribuții este ușurința în utilizare. Distribuția conține toate aplicațiile de care un utilizator are nevoie și realizează multe dintre configurări în mod automat. Un avantaj mare pe care îl oferă este suportul disponibil pe forumul oficial.

Nivelul de adopție al acestui sistem de operare a crescut ușor odată cu distribuirea acestuia preinstalat pe calculatoare. Prima inițiativă în acest sens a avut-o unul dintre cel mai mari producători de echipamente IT care, în mai 2007, a anunțat că va scoate pe piață calculatoare desktop și laptop-uri care au preinstalat Ubuntu. În prezent, un mare număr de computere desktop, laptop dar și de tip netbook sunt vândute cu Linux preinstalat.

2.1.3 Tipuri de distribuții Linux

În IT există două concepte importante legate de stabilitatea, siguranța și gradul de noutate al aplicațiilor (și nu numai): versiunile stabile (*stable*) și cele cu un grad de noutate ridicat (*bleeding edge*). Aceste două concepte reprezintă două extreme ale modului în care este dezvoltată o aplicație.

Versiunile stabile (*stable*) reprezintă versiuni ale programelor care au fost testate un timp îndelungat, ale căror bug-uri au fost rezolvate, care nu prezintă probleme de incompatibilitate și care au un grad de siguranță ridicat. Într-o versiune stabilă este destul de improbabil să apară breșe de securitate. Aceste versiuni apar rar deoarece este nevoie de timp pentru a crea stabilitatea necesară.

Dezavantajul versiunilor stabile este acela că aplicațiile nu prezintă niciun grad de noutate. De multe ori pot să dureze ani până când o nouă versiune a unei aplicații să fie introdusă într-o versiune stabilă a unei distribuții Linux.

Versiunile stabile sunt folosite în special pentru severe datorită gradului de stabilitate și siguranță oferit.

Versiunile cu un grad de noutate ridicat (*bleeding edge*) reprezintă versiuni recente ale programelor, versiuni care includ ultimele îmbunătățiri și inovații. Aceste versiuni apar destul de des, frecvența lor de apariție fiind în mare măsură influențată de mărimea comunității ce dezvoltă aplicația și de îmbunătățirile aduse.

Dezavantajul versiunilor cu grad de noutate ridicat este acela că prezintă un grad de stabilitate și de siguranță incert, de multe ori scăzut. Datorită testării slabe a aplicațiilor (din lipsă de timp), acestea prezintă deseori găuri de securitate sau probleme de stabilitate.

Versiunile cu grad de noutate ridicat sunt recomandate celor care doresc să aibă cele mai noi tehnologii implementate cu orice cost. Ele nu sunt recomandate pentru servere, ci pentru calculatoare unde stabilitatea și siguranța nu sunt elemente cheie.

Între cele două tipuri de aplicați prezentate există numeroase niveluri de compromis. Alegerea unui astfel de nivel pentru un sistem de operare (mai aproape de stabilitate sau mai aproape de cele mai noi tehnologii) trebuie făcută ținând cont de scopul calculatorului pe care va fi instalat și de gradul de securitate și de stabilitate pe care trebuie să îl prezinte acesta.

Distribuțiile au moduri diferite de tratare a conceptelor *stable* și *bleeding-edge*.

Spre exemplu, distribuția Debian poate fi instalată în 3 versiuni:

- *stable* – versiune stabilă, cu aplicații testate mult timp
- *testing* – versiune de testare, după o perioadă de timp devine versiunea *stable*
- *unstable* – versiune care incorporează programele în versiunile lor *bleeding-edge*, devine versiunea testing după o perioadă

Ubuntu are un ciclu de dezvoltare de 6 luni (în engleză *release cycle* – ciclu de lansare). Acest lucru presupune dezvoltarea distribuției timp de 6 luni și lansarea unei noi versiuni după această perioadă. În prezent, Ubuntu lansează distribuții în aprilie (numărul X.04) și în octombrie (numărul X.10) – X reprezintă ultimele două cifre din an (pentru 2009 – 9, 2010 – 10 etc.).

Pe lângă aceste versiuni, Ubuntu are și o versiune LTS (*Long Term Support*) cu un ciclu de dezvoltare de 2 ani, versiune pentru care este oferit suport timp de 3 ani pentru sisteme desktop (5 ani pentru sisteme server).

Pentru fiecare versiune de Ubuntu/Kubuntu, cele mai utilizate trei variante de instalare sunt:

- **Instalarea server** este mai apropiată de Debian și de stabilitatea acestuia. Este recomandată pentru calculatoare care trebuie să ofere stabilitate și siguranță.
- **Instalarea desktop** (cea mai uzuală) permite utilizatorului să instaleze varianta desktop a sistemului de operare folosind un asistent în mod grafic. Pentru o bună funcționare, se recomandă utilizarea unui sistem cu cel putin 384MB de RAM.
- **Instalarea alternate** permite utilizatorului să instaleze varianta desktop, însă asistentul de instalare nu va mai fi unul în mod grafic ci unul în mod text, bazat pe asistentul de instalare al distribuției Debian. Din acest motiv varianta alternate poate fi instalată și pe sisteme cu mai puțin de 384 MB de RAM. Un alt avantaj al variantei alternate este acela că permitea realizarea unei partiționări avansate a hard-disk-ului, oferind suport pentru utilizarea RAID[1] sau LVM[2] (ambele tehnologii permit realizarea de partiții care includ spațiul de pe mai multe hard-disk-uri).

Pe lângă aceste versiune mai există și versiunea *Netbook Remix* (pentru calculatoare de tip *netbook*).

2.2 Instalarea Linux

2.2.1 Alegerea distribuției Linux

După cum a fost menționat și în secțiunea 2.1, există mai multe tipuri de distribuții. Câteva criterii de alegere a unei distribuții au fost menționate anterior.

Principalele criterii folosite în alegerea distribuțiilor sunt:

- nivelul de stabilitate și gradul de noutate al aplicațiilor (în relație cu nivelul de stabilitate, după cum a fost menționat anterior)
- suportul oferit de comunitate (sau de compania care dezvoltă distribuția)
- familiaritatea cu anumite componente ale sistemului, în principal sistemul de pachete (acesta nu reprezintă un criteriu foarte important pentru noii utilizatori)
- suportul pentru hardware-ul disponibil în calculator – distribuțiile pot fi disponibile pentru platforme diferite (procesoare diferite) și pot oferi acces facil la driver-e pentru hardware variat
- resursele disponibile pe calculator – în cazul în care resursele calculatorului pe care va instala Linux sunt scăzute, se recomandă căutarea unei distribuții cu cerințe scăzute

[1] Redundant Array of Inexpensive Disks
[2] Logical Volume Manager

- destinația sistemului de operare – în cazul în care se urmărește utilizarea Linux pentru o anumită activitate/sarcină se recomandă instalarea unei distribuții special concepute (în cazul în care există) – (vezi secțiunea 2.1.1).

Distribuția aleasă ca suport pentru această carte este *Kubuntu 9.10 Desktop*[1], distribuție derivată din Ubuntu, care:

- oferă un nivel acceptabil de stabilitate raportat la nivelul de noutate al pachetelor
- are o comunitate foarte largă, formată din toți utilizatorii de Ubuntu și derivate ale acestuia
- oferă suport facil pentru hardware proprietar, hardware care nu are driver-e open-source disponibile implicit în distribuțiile Linux; majoritatea dispozitivelor comune sunt suportate
- poate funcționa bine pe un sistem desktop actual, dat fiind cerințele acesteia[2]
- sistemul pe care va fi instalat va fi utilizat ca desktop

2.2.2 Pregătirea sistemului

Verificarea cerințelor sistemului. Spațiul de swap

Fiecare sistem de operare (și implicit distribuție Linux) este însoțit de o serie de cerințe minime hardware. Acestea sunt recomandări care, în cazul în care sunt îndeplinite, asigură buna funcționare a sistemului de operare. În anumite situații, programele care se ocupa de instalarea sistemului de operare vor detecta neîndeplinirea cerințelor minime hardware și vor refuza contiuarea procesului de instalare.

Distribuția aleasă în această carte (Kubuntu 9.10) are următoarele cerințe hardware (recomandări, în paranteză se regăsesc cerințele minime):

- procesor x86 700 MHz (min: 300 MHz)
- 384 MB RAM (min: 64 MB RAM – la instalarea fără mediu grafic)
- 8 GB spațiu pe HDD (min: 4 GB)
- placă video capabilă să afișeze rezoluția 1024x768 (min: 640x480)
- pentru instalare și actualizare a sistemului – conexiune la Internet (min: CD-ROM sau placă de rețea)

Printre principalele probleme ce țin de utilizarea facilă a unui sistem de operare o reprezintă lipsa memoriei RAM. Sistemul de operare folosește pentru stocarea datelor temporare memoria RAM. Aceasta are avantajul unui timp de răspuns și al unei viteze de citire foarte mari, însă este limitată din punctul de vedere al capacității (sistemele actuale nu au, în mod normal, mai mult de 2GB de memorie RAM).

Atunci când întreaga memorie RAM este umplută cu date, există două soluții alternative. Prima din ele constă în nealocarea de memorie RAM pentru programele

[1] http://www.kubuntu.org/

[2] https://help.ubuntu.com/community/Installation/SystemRequirements

care cer acest lucru. Ca urmare programul respectiv nu va putea funcționa și va genera erori. Atunci când programul în cauză este unul vital pentru funcționarea sistemului de operare, funcționarea întregului sistem de operare poate deveni instabilă.

A doua variantă este aceea în care datele care ar trebui stocate în memoria RAM sunt stocate într-o alta memorie (pe hard-disk sau pe o memorie pe USB). Deoarece memoria RAM este cea mai rapidă memorie din sistem, utilizarea unui alt tip de memorie va atrage după sine penalizări de performanță. Această soluție se numește **swapping** datorită modului în care funcționează: dacă memoria RAM este plină și apar cereri pentru alocarea de noi zone de memorie, o parte din datele aflate deja în memoria RAM sunt mutate pe un alt tip de memorie, în acest fel eliberându-se memorie RAM. În momentul în care datele mutate anterior pe un alt tip de memorie sunt cerute de programe, ele sunt mutate înapoi în RAM (daca nu este suficient loc în acel moment se va elibera o parte din RAM copiind alte date din RAM pe suportul de memorie).

Cea mai uzuală și ieftină soluție de memorie folosită pentru swap este hard-disk-ul (vezi secțiunea 5.6). Sistemele de operare de tip Linux folosesc în general o zonă dedicată pe hard-disk numită **partiție de swap**. Această partiție are în general dimensiuni apropiate de cele ale memoriei RAM (de ordinul 1-2 GB). Windows-ul și sistemul de operare Mac OS X folosesc pentru swap fișiere a căror dimensiune se poate extinde sau micșora în funcție de necesități.

Partiționare

O **partiție** reprezintă o diviziune logică a unui hard-disk. Prin partiționare un hard-disk este împărțit în mai multe partiții. Pe un hard-disk poate exista și o singură partiție.

Un hard-disk este **nepartiționat** atunci când pe el nu există nicio partiție.

Partiționarea unui hard-disk îl face pe acesta să aibă aceeași comportare ca mai multe hard-disk-uri independente. Diferența între partiții și hard-disk-uri reale constă în faptul că partițiile de pe un hard-disk folosesc aceleași resurse hardware (și, implicit, în cazul unei defecțiuni hardware a hard-disk-ului, datele de pe toate partițiile pot deveni inaccesibile). În plus partițiile pot fi redimensionate după ce au fost create, în schimb discurile au o dimensiune fixă.

Într-o partiție se poate crea un sistem de fișiere pentru stocarea fișierelor sau partiția poate fi utilizată în scopuri speciale, cum sunt de exemplu partițiile de swap. Partițiile de swap nu folosesc un sistem de fișiere ca NTFS, FAT32 sau ext3. Sistemul de operare are metode proprii de a organiza datele în ele și de a le face ușor de accesat (vezi capitolul 4).

Avantajele utilizării de partiții sunt următoarele:

- posibilitatea instalării mai multor sisteme de operare pe același hard-disk – se poate alege ulterior, la pornirea sistemului de calcul, care din ele va fi folosit (vezi secțiunea 6.2);
- posibilitatea de a separa datele personale de datele folosite de sistemul de operare și de programe – dacă datele personale ale utilizatorilor se găsesc pe o

altă partiție, sistemul de operare nu mai poate fi pornit și este necesară reinstalarea lui, datele nu vor fi afectate în niciun fel;

- se poate utiliza același spațiu de swap între mai multe sisteme de operare de tip Linux instalate pe același sistem de calcul (folosind aceeași partiție de swap pentru toate).

Tipuri de partiții

Primul sector fizic de pe un hard-disk poartă numele **Master Boot Record (MBR)** (vezi secțiunea 6.2.1). Acest sector are 512 octeți și conține informații importante despre modul în care este partiționat hard-disk-ul.

Dacă acest sector este șters sau suprascris toate informațiile de pe hard-disk se pierd.[1]

În cadrul MBR se găsesc informații despre fiecare partiție de pe hard-disk. Aceste informații sunt stocate într-o tabelă de partiții primare care are doar patru intrări. Prin urmare MBR-ul poate reține informații pentru doar patru partiții primare. Una dintre aceste patru partiții va fi marcată ca activă și de pe ea se va încerca pornirea sistemului de operare.

O **partiție primară** poate conține doar un singur sistem de fișiere. Pentru a elimina limitarea pe care o prezintă MBR-ul (de a putea reține doar patru partiții primare) a fost creată **partiția extinsă**. Acest tip de partiție ocupă tot o intrare în tabela de partiții din MBR însă poate conține la rândul său mai multe **partiții logice**, fiecare partiție având asociat un sistem de fișiere. Numărul de partiții logice incluse într-o partiție extinsă este limitat doar de dimensiunile partiției extinse și de structura de date folosită pentru a reține informațiile despre partițiile logice.

Unele sisteme de operare trebuie să fie instalate pe partiții primare pentru a putea porni. În această categorie intră MS-DOS și toate versiunile de Windows care depind de el (Windows 3.x, Windows 95, Windows 98, Windows Millenium), precum și BSD.

2.2.3 Live CD

Pentru Ubuntu 9.10 (și derivatele precum Kubuntu) varianta grafică de instalare este disponibilă sub forma unui Live CD.

Un **Live CD** (sau Live DVD) este un CD (DVD) care conține un sistem de operare care poate fi pornit direct (boot-abil), fără a avea nevoie de instalarea pe un hard-disk sau de prezența acestuia.

Există și Live USB-uri, dispozitive USB boot-abile cu sisteme de operare care au, de multe ori, adăugată și posibilitatea scrierii informațiilor pe USB (comparativ cu Live CD/DVD-urile unde informațiile se reține implicit doar în memorie).

[1] chiar dacă informațiile din MBR se pierd, folosind utilitare specializate se pot recupera datele de pe hard-disk prin recunoașterea tiparelor caracteristice sistemelor de fișiere

Termenul "Live" vine de la faptul că pe un singur disc se găsesc toate fișierele necesare funcționării unui sistem de operare, fără a fi necesară copierea unui fișier pe hard-disk înaintea pornirii sistemului de operare.

Astfel un Live CD nu modifică în niciun fel sistemul de operare instalat pe hard-disk, excepție făcând situațiile când utilizatorul dorește în mod expres acest lucru. Un sistem de operare prezent pe un Live CD este totuși unul normal, astfel încât are nevoie de existența unui suport pentru a putea salva informații de stare în fișiere etc.

> Pentru a rezolva problema lucrului cu fișiere, în condițiile în care în mod implicit nu se lucrează cu un hard-disk, un Live CD folosește un **ramdisk** (o porțiune din memoria RAM) precum un hard-disk.

Toate datele care se scriu în *ramdisk* sunt accesibile pe toată perioada în care sistemul de operare rulează. Atunci când acesta este oprit, datele din RAM se pierd și, implicit, tot conținutul *ramdisk*-ului este șters. De aceea, la oprirea unui sistem pornit de pe Live CD, sistemul de calcul revine la starea anterioară pornirii.

Utilizarea unei porțiuni din memoria RAM pe post de hard-disk oferă avantajul unui timp de acces și a unei rate de transfer mai bune decât în cazul unui disk. Dezavantajul este acela că aplicațiile vor avea disponibilă o cantitate mai mica de RAM. Pentru a putea rula în condiții normale un Live CD are nevoie de 512MB de memorie RAM, minimul necesar fiind 256MB.

Așadar un Live CD oferă posibilitatea testării și utilizării unui sistem de operare în condiții de deplină siguranță pentru software-ul instalat pe sistemul de calcul. De multe ori, Live CD-urile sunt folosite pentru recuperarea datelor aflate pe o partiție ce nu mai poate fi accesată din sistemul de operare instalat pe hard-disk (spre exemplu atunci când sistemul de operare nu mai porneste).

2.2.4 Instalare Kubuntu

Kubuntu este o distribuție derivată din Ubuntu, distribuție care folosește ca desktop environment KDE în loc de GNOME (vezi secțiunea 13.7). Kubuntu este parte a proiectului Ubuntu și folosește aceeași arhitectură, aceleași repository-uri și aceleași pachete ca și Ubuntu (vezi secțiunea 3.2).

Pentru a instala Kubuntu (și în general orice distribuție Linux) pe hard-disk este necesară crearea a cel puțin două partiții: una pe care va fi instalat sistemul de operare și una pentru swap. Acest lucru poate fi realizat înaintea instalării propriu-zise (folosind programe specializate în administrarea partițiilor cum ar fi *fdisk*, *GParted*, *PQMagic* etc.) sau poate fi realizat în timpul instalării folosind utilitarul de partiționare pus la dispoziție în acest sens de programul asistent de instalare.

Instalarea Kubuntu 9.10 (varianta Desktop) începe prin introducerea Live CD-ului în unitatea optică și repornirea calculatorului (alternativ, se poate realiza instalarea de pe USB stick). În cazul în care BIOS-ul nu este configurat să inițializeze sistemul de operare de pe unitatea optică (USB stick), trebuie realizată această configurare.

După ce calculatorul a inițializat sistemul de operare de pe CD, va fi afișat un ecran ce permite alegerea modului în care va fi folosit Live CD-ul (figura 2.2)

Figura 2.2: Alegerea modului în care va fi folosit Live CD-ul

Prima opțiune din listă va porni Kubuntu folosind configurările implicite. Din mediul grafic pornit se poate alege ulterior pornirea instalării. A doua opțiune (*Install Kubuntu*) va porni direct instalarea. *Check disc for defects* este util pentru verificarea CD-ului. *Memory test* va porni un test al memoriei RAM, util pentru depanarea problemelor hardware atunci când memoria RAM este suspectată că ar fi defectă.

După ce este aleasă prima opțiune sistemul de operare va fi inițializat. Desktp-ul implicit din Kubuntu este prezentat în figura 2.3.

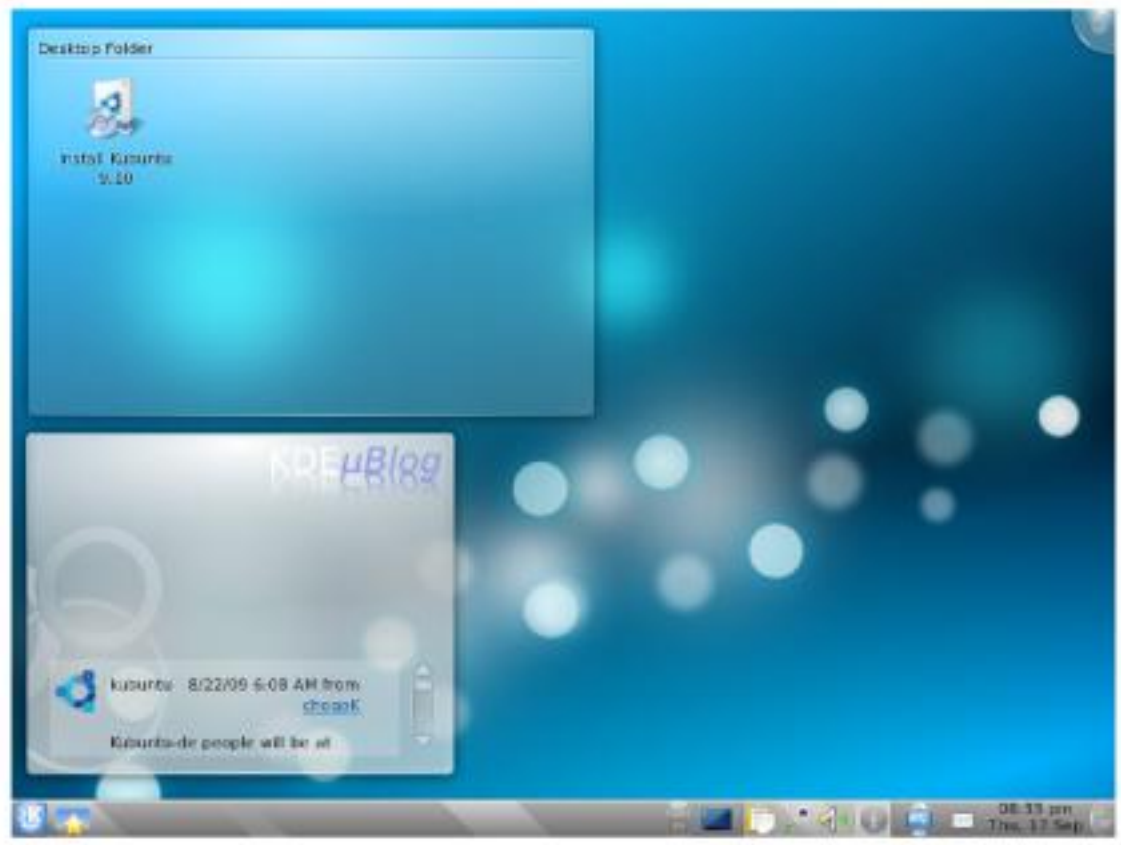


Figura 2.3: Desktop-ul implicit al Live CD-ului Kubuntu

În acest moment Kubuntu poate fi folosit ca Live CD. Pentru a îl instala pe hard-disk, se va accesa pictograma *"Install Kubuntu 9.10"* de pe desktop. Rezultatul va fi pornirea unui asistent de instalare (figura 2.4).

Figura 2.4: Alegerea limbii folosită de asistentul de instalare

Prima etapă constă în alegere limbii folosită de asistentul de instalare. Etapa următoare permite alegerea locului în care se găsește utilizatorul (vezi figura 2.5). Această configurare are efect asupra fusului orar ce va fi stabilit pentru calculator.

Figura 2.5: Alegerea locului în care se află utilizatorul

Pasul următor este reprezentat de alegerea felului în care sunt amplasate tastele pe tastatură (*keyboard layout*) (figura 2.6). După ce se realizează această configurare urmează pasul cel mai important din cadrul instalării: partiționarea hard-disk-ului (vezi figura 2.7)

Discul poate fi șters integral, caz în care datele deja existente vor fi șterse, sau poate fi partiționat manual, alegând dimensiunea partițiilor și tipul lor. În continuare va fi prezentată partiționarea manuală.

Figura 2.6: Alegerea felului în care sunt amplasate tastele pe tastatură

Figura 2.7: Partiționarea manuală a hard disk-ului

În cazul în care discul este nepartiționat este necesară crearea unei noi tabele de partiții apăsând butonul *New Partition table*. Rezultatul va fi afișarea tabelei de partiții, prezentată și în figura 2.8.

În cazul în care exista deja partiții pe disc, acestea vor fi afișate în tabela de partiții. În continuare, se presupune că hard-disk-ul nu este partiționat și se vor crea două partiții: una *ext4* și una pentru *swap*. Pentru a realiza acest lucru se apasă butonul *Add...* din fereastra care prezintă tabela de partiții (figura 2.9).

După cum se observă în figura 2.9, partiția *ext4* este primară, are 4096 MB, este creată la începutul spațiului liber și va fi accesată de sistemul de operare prin directorul / (*mount point*).

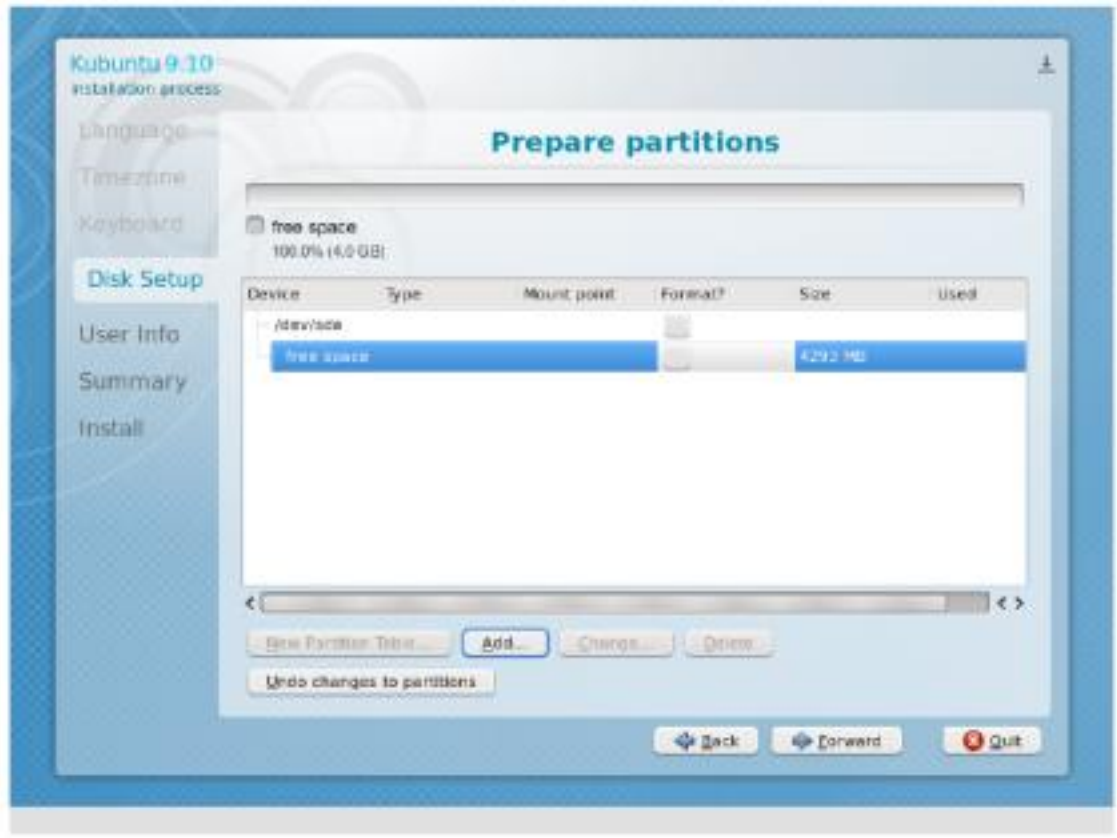


Figura 2.8: Tabela de partiții – inițial discul nu are nicio partiție

Figura 2.9: Crearea unei partiții

Figura 2.10: Crearea unei partiții swap

Partiția de swap (figura 2.10) este tot primară, are 197 MB și este creată la începutul spațiului liber rămas. Ea nu va putea fi accesată printr-un director, ci va fi folosită de sistemul de operare în mod transparent.

Această schemă de partiționare este pur demonstrativă. După crearea celor două partiții, tabela de partiții arată ca în figura 2.11.

După crearea partițiilor, trebuie datele personale ale utilizatorului: numele lui, numele de utilizator dorit, parola și numele stației (*hostname*). Asistentul de instalare mai permite și alegerea modului de autentificare în sistem și dacă directorul cu datele utilizatorului

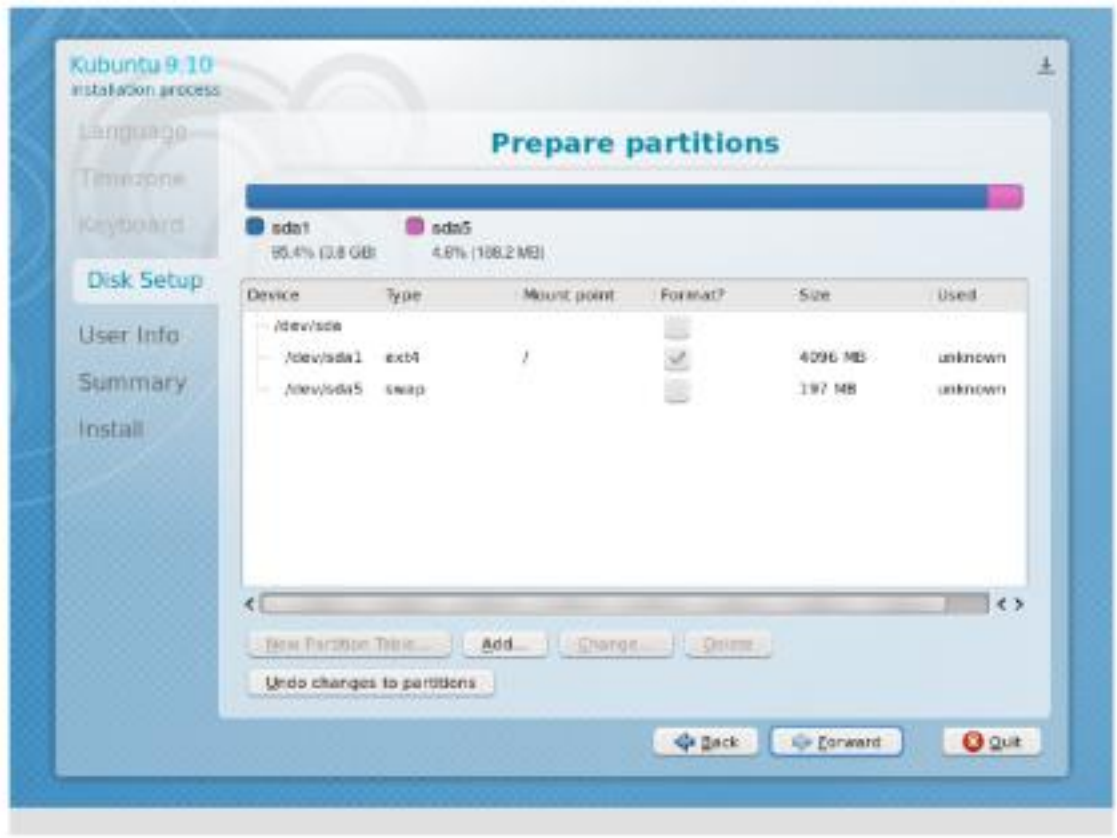


Figura 2.11: Tabela de partiții după ce au fost create cele două partiții

va fi criptat (vezi figura 2.12) (vezi secțiunea 10.5.2).

Figura 2.12: Configurarea datelor personale ale utilizatorului

În pasul următor (ultimul pas al instalării) pot fi revizuite configurările realizate. Apăsând pe butonul *Advanced...* se poate configura unde va fi instalat *bootloader*-ul (vezi secțiunea 6.2) bootloader/GRUB (vezi figura 2.13).

Pentru a începe copierea fișierelor pe hard-disk se apară butonul *Install*. La terminarea instalării va fi afișat mesajul din figura 2.14.

După închiderea asistentului de instalare sistemul de operare va rula în continuare ca Live CD. Pentru a porni sistemul de operare proaspăt instalat, calculatorul trebuie repornit.

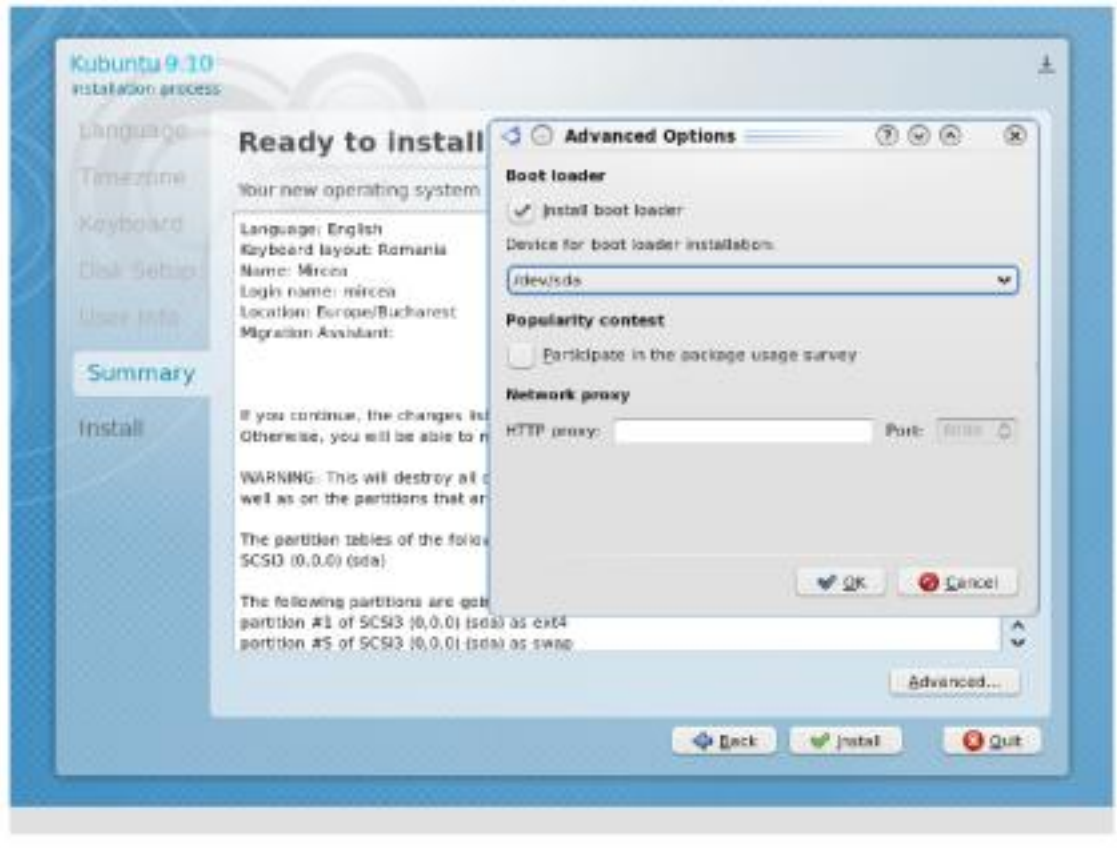


Figura 2.13: Verificarea datelor și alegerea hard-disk-ului pe care va fi instalat GRUB

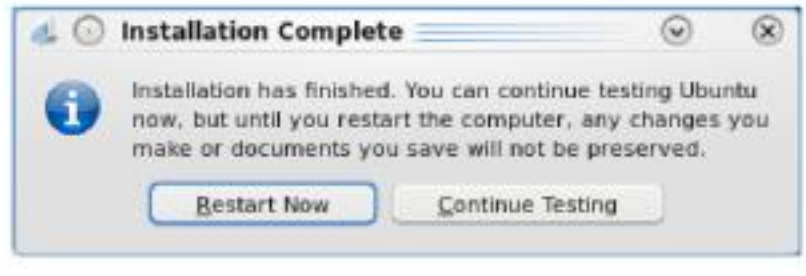


Figura 2.14: Mesajul care marchează instalarea cu succes a sistemului de operare

2.3 Interacțiunea cu sistemul de operare

2.3.1 Interfețele cu utilizatorul

Primele interfețe cu utilizatorul au fost, din motive tehnice, create pe monitoare care afișau doar caractere text. De aceea, posibilitățile acestor interfețe de a crea componente grafice au fost limitate. În această categorie (interfețe care utilizează doar caractere în mod text) intră CLI (*Command Line Interface*) și TUI (*Text User Interface*). În timp, atât monitoarele cât și plăcile video au avansat tehnologic și au apărut astfel afișajele grafice (cu rezoluții mai mari decât cele text). În acest fel a apărut și o nouă clasa de interfețe cu utilizatorul numită GUI (*Graphical User Interface*).

În **CLI**, interacțiunea cu sistemul de operare se bazează pe comenzi scrise de la tastatură. Exemple de CLI pot fi văzute în figura 2.17 și în figura 2.18.

Avantajul acestei metode este acela că pot fi invocate comenzi complexe utilizând secvențe scurte de caractere. Pentru a realiza sarcini echivalente într-un GUI poate fi uneori nevoie și de câteva zeci de click-uri de mouse. Un alt avantaj al CLI este rapiditatea dobândită în utilizare, după ce comenzile au fost reținute (în special dacă este folosit istoricul de comenzi: prin apăsarea tastei `SăgeatăSus` pot fi reexecutate comenzile date anterior). Luând ca exemplu configurarea unei adrese IP (vezi secțiunea 8.2.1) pe o interfață de rețea, diferența de viteză între prima și a zecea oară când se realizează această configurare într-un GUI este mica, datorită numărului mare

de click-uri necesar. În cazul unui CLI, folosind și istoricul de comenzi, diferența de viteză este extrem de mare.

Dezavantajul principal al CLI este curba abruptă de învățare, utilizatorul obișnuindu-se mai greu cu o comandă decât cu reținerea locului în care se află un buton.

Deși de la apariția sa a fost concurat de TUI și GUI, CLI rămâne metoda de control al sistemului de operare preferată de utilizatorii avansați datorită eficienței în utilizare.

În **TUI**, interacțiunea cu sistemul de operare se realizează prin intermediul ferestrelor, meniurilor, butoanelor și mouse-ului. Pentru desenarea interfeței sunt utilizate caractere disponibile pe orice terminal text (figura 2.15).

Figura 2.15: Text User Interface

Acest tip de interfață se apropie de modul în care arată o interfață grafică însă, datorită faptului că folosește caractere text, rezoluția ecranului este mai mică.

GUI este un sistem de interacțiune cu utilizatorul de tipul WIMP (*window, icon, menu, pointing device*) (figura 2.16).

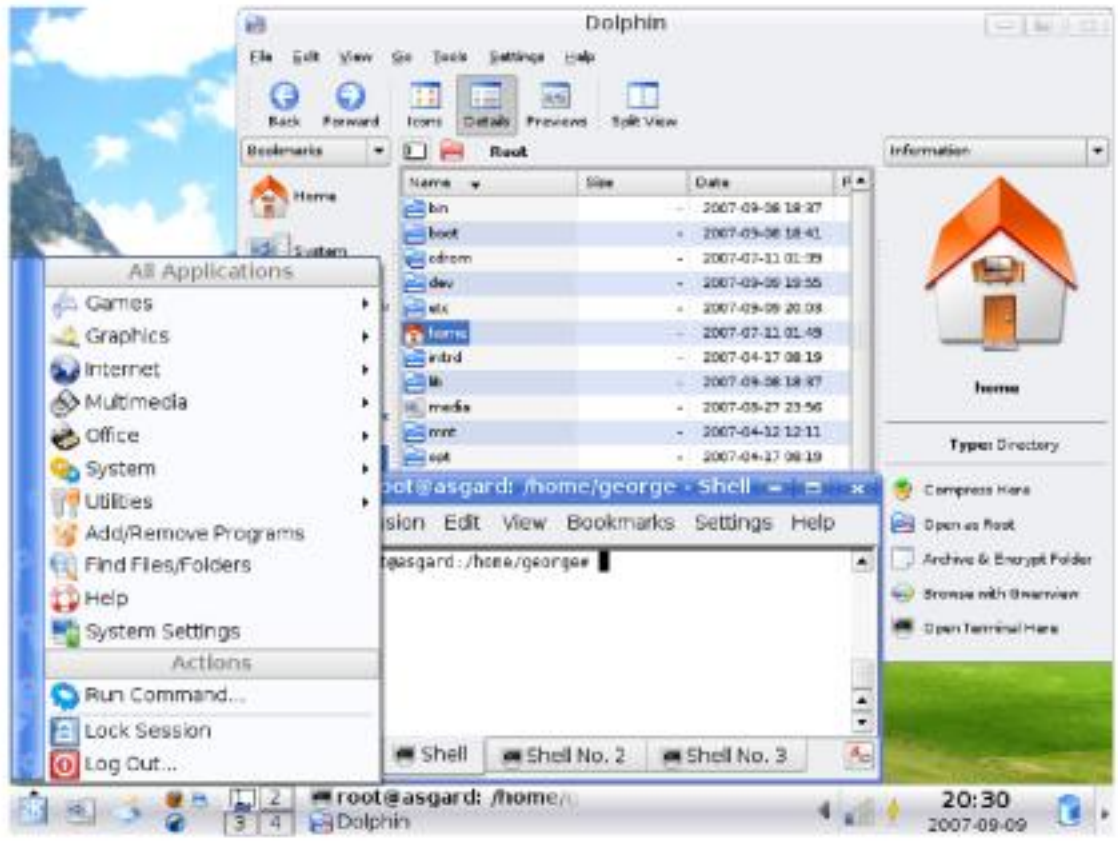


Figura 2.16: Graphical User Interface

Pentru desenarea interfeței GUI sunt folosite moduri grafice de rezoluție înaltă, cum ar fi VGA=640x480, SVGA=800x600, 1024x768, 1280x1024, 1600x1200 etc.

2.3.2 Interfața în linie de comandă (CLI)

Consola

> O **consolă** este un tip de echipament care permite introducerea și afișarea de text folosit în administrarea și utilizarea sistemului de operare.

Inițial consolele erau formate dintr-un terminal (un monitor ce avea atașată o tastatură) care se conecta la sistemul de calcul printr-o conexiune serială. Acest tip de console este foarte rar întâlnit în ziua de astăzi.

Pe un calculator obișnuit, monitorul împreună cu tastatură au același rol ca un terminal. Ele permit, în Linux, accesarea a două tipuri de console: *terminalele virtuale* (console în mod text) și *emulatoarele de terminale* (console în mod grafic).

Terminalele virtuale au o rezoluție mică (în general 25 de linii cu 80 de caractere pe linie). Ele pot fi accesate din interfață grafică cu ajutorul tastelor `CTRL+ALT+Fx`, unde Fx reprezintă tastele F1 până la F6 (în mod implicit). Din aceste terminale virtuale se poate reveni în interfață grafică cu ajutorul combinației de taste `(CTRL+)ALT+F7`.

Trecerea dintr-un terminal virtual în altul poate realiza prin trei combinații de taste. Primele două sunt: `CTRL+ALT+Fx` și `ALT+Fx` (unde x este numărul terminalului în care se va trece). A treia combinație de taste este `ALT+Sageată Stânga` sau `ALT+Sageată Dreapta` și permite trecerea dintr-un terminal în cel precedent, respectiv în cel următor. De exemplu trecerea din terminalul 4 în terminalul 3 se poate realiza cu `CTRL+ALT+F3` sau `ALT+F3` sau `ALT+Sageată Stânga`.

Valorile tastelor `F` prezentate sunt cele implicite pe majoritatea distribuțiilor. Aceste valori pot fi schimbate. De exemplu, interfața grafică poate fi plasată pe terminalul trei sau pe terminalul cinci; numărul de terminale accesibile poate fi șase (valoarea implicită) sau zece (figura 2.17).

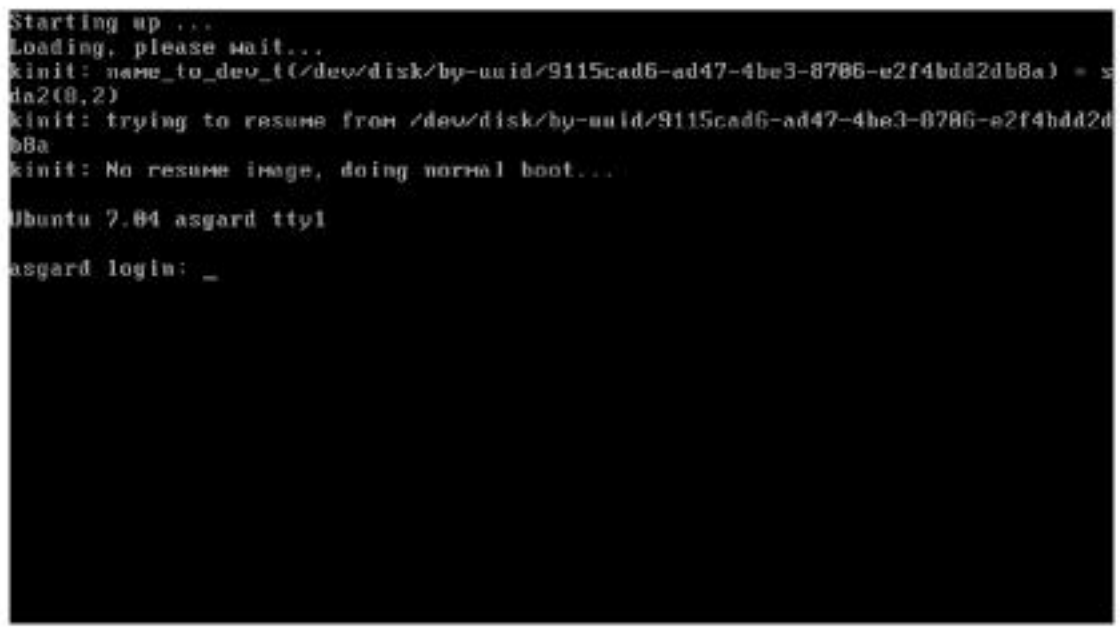


Figura 2.17: Terminal virtual (CLI)

Consolele în mod grafic reprezintă **emulatoare de terminal** (figura 2.18). Dimensiunea lor pot fi extinsă peste cea de 25x80 a terminalelor virtuale. De asemenea, un utilizator poate porni un număr nelimitat de astfel de emulatoare. Cele mai cunoscute emulatoare sunt **`konsole`** (pentru KDE), **`gnome-terminal`** (pentru GNOME) și **`xterm`**.

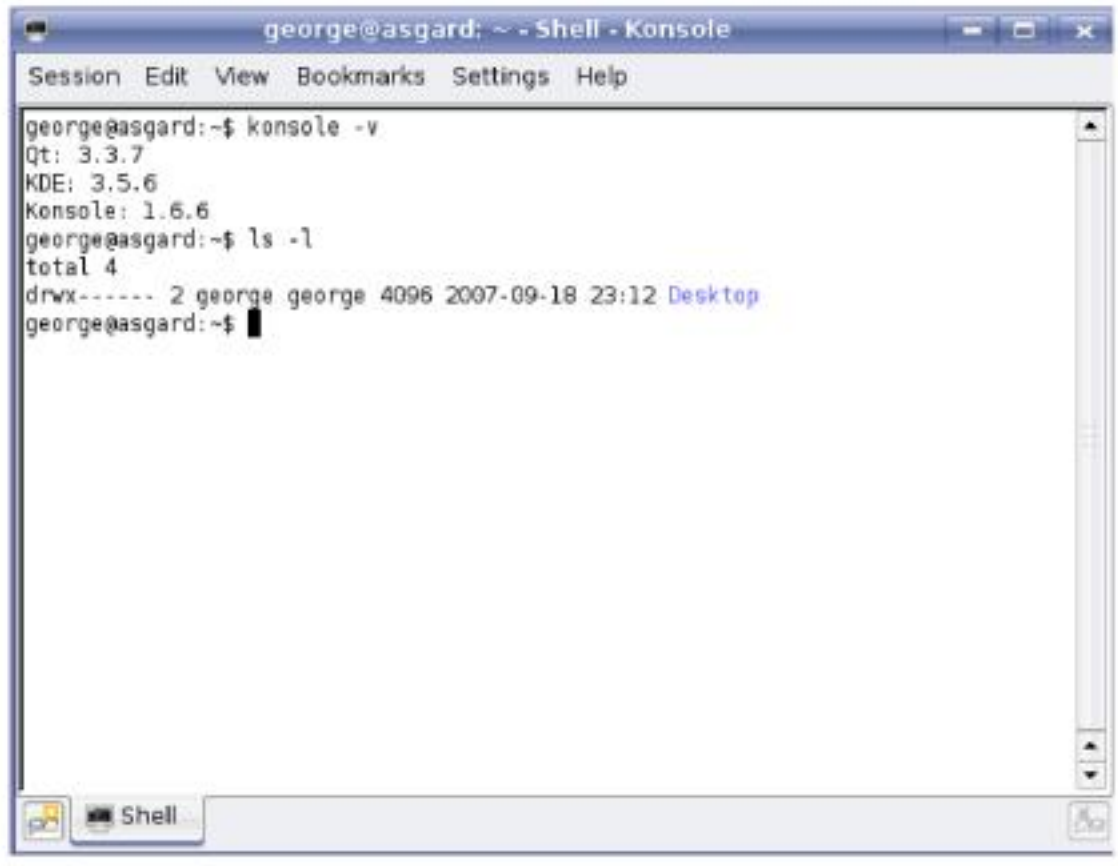


Figura 2.18: Emulator consolă (CLI)

Atunci când un utilizator accesează un terminal virtual el trebuie să se autentifice folosind un nume de utilizator și o parolă. La pornirea unui emulator de terminal utilizatorul nu mai trebuie să se autentifice, deoarece autentificarea s-a produs înainte de pornirea interfeței grafice. De asemenea, la pornirea unui emulator de terminal utilizatorul curent va fi cel care este autentificat în interfața grafică.

În ultimii ani, îndepărtarea de linia de comandă a sistemului de operare Linux este din ce în ce mai accentuată în detrimentul utilizării interfeței grafice, mai intuitive și cu facilități mult mai avansate. Cu toate acestea, interfața în linia de comandă rămâne o componentă esențială în configurarea rapidă a unui sistem și a administrării de la distanță. Interfața în linia de comandă permite toate operațiile care pot fi realizate și din interfața grafică.

Shell

În IT, denumirea de *shell* are mai multe sensuri. Unul dintre acestea este sensul de componentă software care realizează interfața cu utilizatorul. Această interfață poate fi una CLI, caz în care *shell*-ul este un interpretor de comenzi, sau una GUI, caz în care *shell*-ul este o interfață grafică dintre utilizator și resursele sistemului de operare (cum ar fi Explorer în Windows).

Comenzile Linux sunt transmise la o consolă. Consola (indiferent de tipul ei – emulator de terminal sau terminal virtual) rulează un shell, shell care permite introducerea

comenzilor. Există mai multe aplicații shell de interpretare a comenzilor în Linux, cea mai uzuală fiind denumită **bash** (Bourne Again SHell).

Prompt-ul

Shell-ul (interpretorul) de comenzi oferă utilizatorului un prompt. Acesta are un format de genul:

```
1 username@localhost:~$
```

La acest prompt, utilizatorul poate introduce comenzi. `username` reprezintă numele utilizatorului, iar `localhost` este numele stației (*hostname*). Simbolul `~` indică directorul curent, home-ul utilizatorului `george`[1]. Simbolul `"$"` marchează terminarea prompt-ului și începerea zonei de introducere a comenzilor. De exemplu, în cazul următor:

```
1 george@asgard:~$
```

`george` este numele utilizatorului și `asgard` este numele stației.

Pentru mai multe informații despre interacțiunea cu shell-ul consultați secțiunea 12.2.

Comenzi

Comenzile sunt cuvinte cheie care se introduc la consolă (și interpretate de interpretorul de comenzi), folosite pentru a configura sistemul de operare sau pentru a obține informații de la acesta.

Comenzile pot fi simple sau pot avea parametri. O **comandă simplă** este o comandă care conține un singur cuvânt cheie suficient pentru a executa o acțiune. Majoritatea comenzilor sunt însă **comenzi compuse** și pot primi unul sau mai mulți parametri care pot afecta rezultatul comenzii. Separarea între numele comenzii și parametru sau între doi parametri se realizează prin intermediul caracterului "spațiu". De multe ori, primul parametru (sau și alți parametri, în funcție de caz) este marcat prin caracterul "-".

Mai jos este prezentat un exemplu de comandă simplă și apoi aceeași comandă cu parametri. Comenzile au de obicei rezultate, rezultate care sunt afișate de către interpretorul de comenzi (shell) pe consola utilizatorului. În exemplul nostru, fiecare comandă are un rezultat care este afișat.

```
1  george@asgard:~$ ps
2    PID TTY          TIME CMD
3  13391 pts/0    00:00:00 bash
4  13666 pts/0    00:00:00 ps
5
6  george@asgard:~$ ps -o uid,pid,cmd
7    UID   PID CMD
8   1000 13391 -bash
9   1000 13669 ps -o uid,pid,cmd
10
11 george@asgard:~$
```

[1] detalii despre ierarhia sistemului de fișiere se găsesc în secțiunea 4.1.2

Setul cel mai important de comenzi este cel care interacționează cu sistemul de fișiere. Aceste comenzi sunt fundamentale pentru interacțiunea cu sistemul și rularea altor comenzi importante.

Căutarea de ajutor

Daca se dorește căutarea de informații în legătură cu o comandă (parametri, date de ieșire, funcționalitate etc.) distribuțiile de Linux pun la dispoziție mai multe opțiuni:

- Transmiterea parametrului `--help` unei comenzi pentru a afișa un sumar al parametrilor posibili ai acelei comenzi. De exemplu:

```
1  george@asgard:~$ cp --help
2  Usage: cp [OPTION]... [-T] SOURCE DEST
3    or:  cp [OPTION]... SOURCE... DIRECTORY
4    or:  cp [OPTION]... -t DIRECTORY SOURCE...
5
6  Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.
7
8
9  Mandatory arguments to long options are mandatory for short options
   too.
10
11   -a, --archive                same as -dpR
12       --backup[=CONTROL]       make a backup of each existing
   destination file
13   -b                           like --backup but does not accept an
   argument
14       --copy-contents          copy contents of special files when
   recursive
15   -d                           same as --no-dereference --preserve=link
16 [...]
```

- Comanda **`whatis`** este folosită pentru a afișa o scurtă descriere a unei comenzi:

```
1  george@asgard:~$ whatis ls
2  ls (1)               - list directory contents
3  george@asgard:~$ whatis pwd
4  pwd (1)              - print name of current/working directory
5  george@asgard:~$ whatis cp
6  cp (1)               - copy files and directories
7  george@asgard:~$ whatis poweroff
8  poweroff (8)         - stop the system.
```

- Comanda **`apropos`** afișează comenzile care au legătură cu cuvântul transmis ca parametru:

```
1  george@asgard:~$ apropos zip
2  bunzip2 (1)  - a block-sorting file compressor, v1.0.3
3  bzcmp (1)    - compare bzip2 compressed files
4  bzdiff (1)   - compare bzip2 compressed files
5  bzegrep (1)  - search possibly bzip2 compressed files for a regular
   expression
6  bzfgrep (1)  - search possibly bzip2 compressed files for a regular
   expression
7  [...]
```

- Comenzile **man** și **info** sunt utilizate pentru a afișa paginile de documentație complete asociate unei comenzi dorite. Aceste pagini conțin informații detaliate despre opțiunile posibile ale comenzilor și utilizările acestora. Comenzile **man** și **info** primesc ca argument numele comenzii a cărei documentație se dorește a fi afișată:

```
1 george@asgard:~$ man ls
2 Reformatting ls(1), please wait...
3
4 george@asgard:~$ man pwd
5 Reformatting pwd(1), please wait...
6
7 george@asgard:~$ info coreutils ls
8 george@asgard:~$ info coreutils cp
```

Autentificarea și părăsirea sistemului

Autentificarea locală în sistemul de operare se poate realiza folosind un terminal virtual sau se poate realiza în interfață grafică, printr-un *display manager* (vezi secțiunea 13.2.2).

În cazul în care utilizatorul s-a autentificat în interfață grafică, el poate părăsi sistemul de operare (*logout*) tot din mediul grafic, folosind opțiunea *Logout*.

În cazul în care utilizatorul s-a autentificat într-un terminal virtual, el poate părăsi sistemul de operare folosind comanda **logout** sau comanda **exit**.

Comanda **logout** este folosită pentru a părăsi un login shell. Acesta este un shell creat atunci când utilizatorul s-a autentificat în sistem.

Comanda **exit** este folosită pentru a părăsi atât un login shell cât și un shell creat din acest login shell (cu ajutorul comenzii **su** de exemplu).

2.3.3 Oprirea sistemului de calcul

Pentru a opri sau reporni un sistem de calcul se folosește comanda **shutdown**. Această comandă poate fi utilizată doar de către un utilizator cu drepturi administrative (vezi secțiunea 2.4.1). Sintaxa comenzii este:

```
1 shutdown [OPTION]... TIME [MESSAGE]
```

Printre opțiunile folosite în mod uzual se numără: `-r` pentru a reporni stația și `-h` pentru a o opri. `time` reprezintă momentul la care va fi oprită sau repornită stația, iar `message` este un mesaj ce va fi trimis pe toate terminalele legate la sistem.

De exemplu, pentru a opri stația imediat, se rulează comanda:

```
1 root@asgard:/home/george# shutdown -h now
2
3 Broadcast message from george@asgard
4         (/dev/pts/0) at 18:28 ...
5
6
7 The system is going down for halt NOW!
```

Pentru a reporni stația imediat, se rulează comanda:

```
root@asgard:/home/george# shutdown -r now
Broadcast message from george@asgard
        (/dev/pts/4) at 11:04 ...


The system is going down for reboot NOW!
```

Pentru a opri stația la ora 20:00:

```
root@asgard:/home/george# shutdown -r 20:00

Broadcast message from george@asgard
        (/dev/pts/4) at 11:03 ...


The system is going down for reboot in 537 minutes!
```

Pentru a opri stația în 10 minute de la executarea comenzii se folosește comanda:

```
root@asgard:/home/george# shutdown -r +10

Broadcast message from george@asgard
        (/dev/pts/0) at 18:33 ...

The system is going down for reboot in 10 minutes!
```

O altă posibilitate de a reporni stația este utilizarea comenzii **`restart`**. Pentru oprirea stației se pot folosi și comenzile **`halt`** sau **`poweroff`**. Toate aceste comenzi pot fi executate doar daca utilizatorul are cu drepturi administrative.

2.4 Configurări de bază ale SO

2.4.1 Administrarea sistemului Linux

În sistemele de operare Windows până la Windows XP (inclusiv), în timpul procesului de instalare se crea un utilizator cu drept de administrare. De cele mai multe ori, sistemul era folosit cu acest utilizator, situație care putea crea probleme de securitate datorită drepturilor extinse de care beneficia utilizatorul (vezi secțiunea 10.2). În sistemele de operare Linux se face o distincție implicită între drepturile de administrare și drepturile de utilizare a sistemului.

> În Linux, un singur utilizator are drepturi de administrare depline. Acesta este un utilizator special numit **`root`**.

Utilizatorul `root` poate realiza configurări, poate modifica modul în care pornește sistemul de operare, poate acorda drepturi parțiale altor utilizatori etc.

Pentru a putea realiza configurări sau acțiuni pe care doar utilizatorul `root` le poate realiza, un utilizator are trei posibilități.

Prima posibilitate o reprezinta autentificare în sistemul de operare folosind numele de utilizator `root` și parola asociată acestuia. În cele mai multe sisteme de operare folosite

în producție autentificarea ca `root` este dezactivată din motive de siguranță. De aceea această metoda poate să nu funcționeze.

Schimbarea între utilizatori (`su`)

În cazul în care utilizatorul este deja autentificat în sistem, există posibilitatea schimbării utilizatorulu (**a doua posibilitate** de obținere a drepturilor administrative). Din *shell*-ul în care se găsește deja, utilizatorul poate să pornească un nou *shell* în care să se autentifice ca un alt utilizator. Atunci când părăsește *shell*-ul nou creat (de exemplu prin comanda **`exit`**) el va reveni în *shell*-ul inițial.

Pornirea unui nou *shell* în care se realizeaze autentificarea ca alt utilizator se face cu ajutorul comenzii **`su`** (*switch user*). Această comandă poate primi ca parametru numele utilizatorului în contul căruia se va face autentificarea:

```
1 george@asgard:~$ su uso
2 Password:
3
4 uso@asgard:/home/george$ exit
5 exit
6
7 george@asgard:~$
```


Din motive de securitate, caracterele tastate în dreptul parolei nu vor fi afișate.

În exemplul anterior, utilizatorul autentificat în sistem este `george`. Utilizatorul `george` schimbă utilizatorul în `uso` folosind comanda **`su uso`** și se autentifică utilizând parola utilizatorului `uso`. Utilizatorul părăsește shell-ul nou folosind comanda `exit`, moment în care se observă revenirea acestuia în shell-ul inițial.

În cazul în care comanda **`su`** nu primește niciun parametru, implicit se va considera ca se dorește autentificarea ca utilizator `root`.

Ca root, execuția comenzii `su uso` va permite autentificarea ca utilizator `uso` fără introducerea parolei pentru acesta.

Pentru a părăsi un *shell* creat cu **`su`** nu se poate folosi comanda **`logout`**, deoarece sesiunea autentificată în sistem (login shell) aparține tot utilizatorului inițial (utilizatorul inițial fiind `george` în exemplul de mai jos) și nu noului utilizator (noul utilizator fiind `uso` în exemplul de mai jos):

```
1  george@asgard:~$ su uso
2  Password:
3
4  uso@asgard:/home/george$ logout
5  bash: logout: not login shell: use 'exit'
6
7  uso@asgard:/home/george$ exit
8  exit
9
10 george@asgard:~$
```

Ubuntu (și Kubuntu) au dezactivat în mod implicit contul de `root`, nefiind configurată nicio parolă pentru acesta. De aceea, autentificarea ca `root` atât la *prompt*-ul de login în sistem cât și cu **`su`** va eșua.

Execuția comenzilor cu drepturi administrative (sudo)

În cazul în care este dezactivată autentificarea ca `root`, **a treia posibilitate** de obținere a drepturilor administrative o reprezintă utilizarea comenzii **sudo**. Această comandă permite executarea unui program ca alt utilizator. Spre deosebire de **su**, care cerea parola utilizatorului al cărui cont va fi utilizat, **sudo** cere parola utilizatorului curent[1]. Din acest motiv comanda **sudo su** va avea succes (este cerută parola utilizatorului curent, nu cea a utilizatorului `root`, după care comanda **su** este executata ca `root`):

```
1 george@asgard:~$ sudo su
2 Password:
3
4 root@asgard:/home/george#
```

Atunci când utilizatorul `root` folosește comanda **sudo** pentru a porni un *shell* în care să se autentifice ca alt utilizator, nu i se va cere nicio parolă.

2.4.2 Asigurarea conectivității la Internet

Una dintre primele configurări ce trebuie realizată pe sistemul de operare instalat este asigurarea conectivității la Internet. Kubuntu este o distribuție de Linux care se bazează pe pachete și necesită o conexiune la Internet pentru a putea copia și instala versiuni mai recente ale acestora, realizând în acest fel actualizarea întregului sistem de operare.

Pentru asigurarea conectivității la Internet există mai multe alternative. Cea mai simplă variantă este aceea în care toți parametrii de rețea (vezi secțiunea 8.2) (adresă IP, mască de rețea, gateway, server DNS) sunt configurați în mod automat folosind DHCP[2]. Această metodă este cea încercată implicit de către sistemul de operare. Dacă aceasta reușește, utilizatorul are acces la Internet, nemaifiind necesară o altă configurare.

Dacă încercarea de configurare folosind DHCP eșuează, utilizatorul va trebui să configureze manual parametrii de rețea. Acest lucru se poate realiza *temporar* (efectele configurărilor vor dura până la următoarea pornire a sistemului de operare sau până la o nouă configurare) sau *permanent* (efectele configurărilor se vor păstra și după repornirea sistemului de operare).

Configurările temporare pentru adresa IP, masca de rețea și gateway se realizează folosind comenzile **ifconfig** și **route**. Pentru a putea folosi aceste comenzi utilizatorul trebuie să obțină drept de administrare. Pentru mai multe informații legate de configurarea rețelei consultați capitolul 8.

2.4.3 Actualizarea sistemului și a pachetelor

Printre primele acțiuni care se realizeaza imediat după instalarea unui sistem de operare se regăsește actualizarea acestuia. Într-un sistem bazat pe Debian (precum Ubuntu și Kubuntu), acest lucru se face prin execuția următoarelor comenzi ca `root`:

[1] acest comportament este dependent de distribuție – în Ubuntu și în distribuțiile derivate, comanda **sudo** cere parola utilizatorului curent

[2] Dynamic Host Configuration Protocol

```
root@asgard:~# apt-get update

root@asgard:~# apt-get dist-upgrade
```

Prima comanda actualizeaza lista de pachete cunoscută de către managerul de pachete, iar a doua comandă actualizează pachetele instalate pe baza informațiilor adunate de prima comandă. Pentru mai multe informații despre gestiunea pachetelor consultați secțiunea 3.2.

2.4.4 Administrarea utilizatorilor și a grupurilor de utilizatori

Majoritatea programelor de instalare a distribuțiilor Linux permit celui care realizează procesul de instalare să adauge direct utilizatori non-root în sistem, fără a fi nevoie de configurări ulterioare.

În cazul în care mai trebuie adăugați utilizatori sau grupuri, administratorul are la dispoziție comenzile **`adduser`** și **`addgroup`** pe sisteme Debian (și implicit Ubuntu/Kubuntu). **`adduser`** permite adăugarea în mod interactiv de utilizatori.

Dacă se dorește lucrul mai avansat cu utilizatori și grupuri (precum ștergerea, modificarea utilizatorilor/grupurilor) consultați secțiunea 3.1.

2.5 Studii de caz

2.5.1 Interoperabilitatea sistemului de fișiere între Linux și Windows

Atunci când pe un calculator este instalat și un sistem de operare Windows și unul Linux se pune de cele mai multe ori problema partajării de fișiere între cele două. Ca exemplu concret:

- cum se pot accesa din Linux (pentru scriere și citire) datele aflate în Windows
- cum se pot accesa din Windows datele aflate în Linux (pentru citire și scriere).

Soluția la această problemă se bazează pe ideea că pentru datele aflate pe cel puțin o partiție se va asigura suportul de citire și de scriere atât din Linux cât și din Windows. În acest sens există trei variante.

Prima dintre ele constă în utilizarea unei partiții FAT32 pentru a stoca datele ce se doresc a fi partajate. Avantajul acestei soluții este faptul că, atât în Windows cât și în Linux, există suport nativ pentru sistemul de fișiere FAT. Dezavantajele majore sunt limitarea dimensiunii unei partiții FAT32 la maxim 32GB și limitarea dimensiunii maxime a unui fișier la 4GB (ca atare pe o astfel de partiție nu se poate salva imaginea unui disc DVD de exemplu).

A doua soluție o reprezintă utilizarea unei partiții ext2 sau ext3 pentru stocarea datelor partajate. Această soluție nu prezintă limitările partiției FAT, însă Windows nu oferă suport nativ pentru acest tip de partiții. Pentru a putea accesa din Windows o partiție

ext2 sau *ext3* trebuie instalat un driver special. Acest driver se numește **Ext2 IFS**și este disponibil online[1].

Acest driver oferă suport de scriere și citire pentru sistemul de fișiere *ext2*. Sistemul de fișiere *ext3* este asemănător cu *ext2*, diferența dintre ele fiind reprezentată de suportul de jurnalizare pe care îl oferă *ext3*. *Ext2 IFS* poate citi și scrie date de pe o partiție *ext3*, însă nu va folosi suportul de jurnalizare pe care îl oferă aceasta.

Ext2 IFS nu funcționeaza cu sistemul de fișiere *ext4*, structurile de date aflate pe disc fiind diferite. Acest lucru trebuie luat în considerare în condițiile în care se dorește utilizarea acestei variante pentru partajare, mai ales când distribuțiile actuale au ca sistem de fișiere implicit *ext4* (vezi și secțiunea 4.7).

A treia variantă o reprezintă accesarea unei partiții NTFS din Linux. În Linux există suport pentru citirea datelor de pe un sistem de fișiere NTFS. Datorită faptului că standardul NTFS este unul închis, dezvoltarea unui driver care să ofere suport de scriere a fost greu de realizat. În momentul de față există însă un driver care oferă acest suport. El se numește *ntfs-3g*[2] și suport pentru acesta este oferit implicit în Ubunut/Kubuntu.

2.5.2 GParted

GParted este o aplicație de partiționare specifică Gnome[3] (figura 2.19). Aceasta poate crea, șterge, redimensiona, muta, verifica și copia partiții.

GParted oferă și o versiune disponibilă sub forma unui Live CD[4]. Aceasta este bazată pe distribuția Gentoo și permite administrarea partițiilor în cazul în care sistemul de operare nu mai pornește. De asemenea permite realizarea unei copii de siguranță a datelor în aceleași condiții.

Cuvinte cheie

- CLI
- consolă
- distribuție Linux
- gestionare utilizatori
- GUI
- Kubuntu
- Live CD
- MBR
- instalare
- mediu grafic
- partiție
- swap
- shell
- shutdown
- su
- sudo
- TUI

[1] http://www.fs-driver.org/

[2] http://www.ntfs-3g.org/

[3] Gnome este un Desktop Environment pentru mediul grafic din Linux – mai multe informații se pot găsi în secțiunea 13.7

[4] http://gparted.sourceforge.net/livecd.php

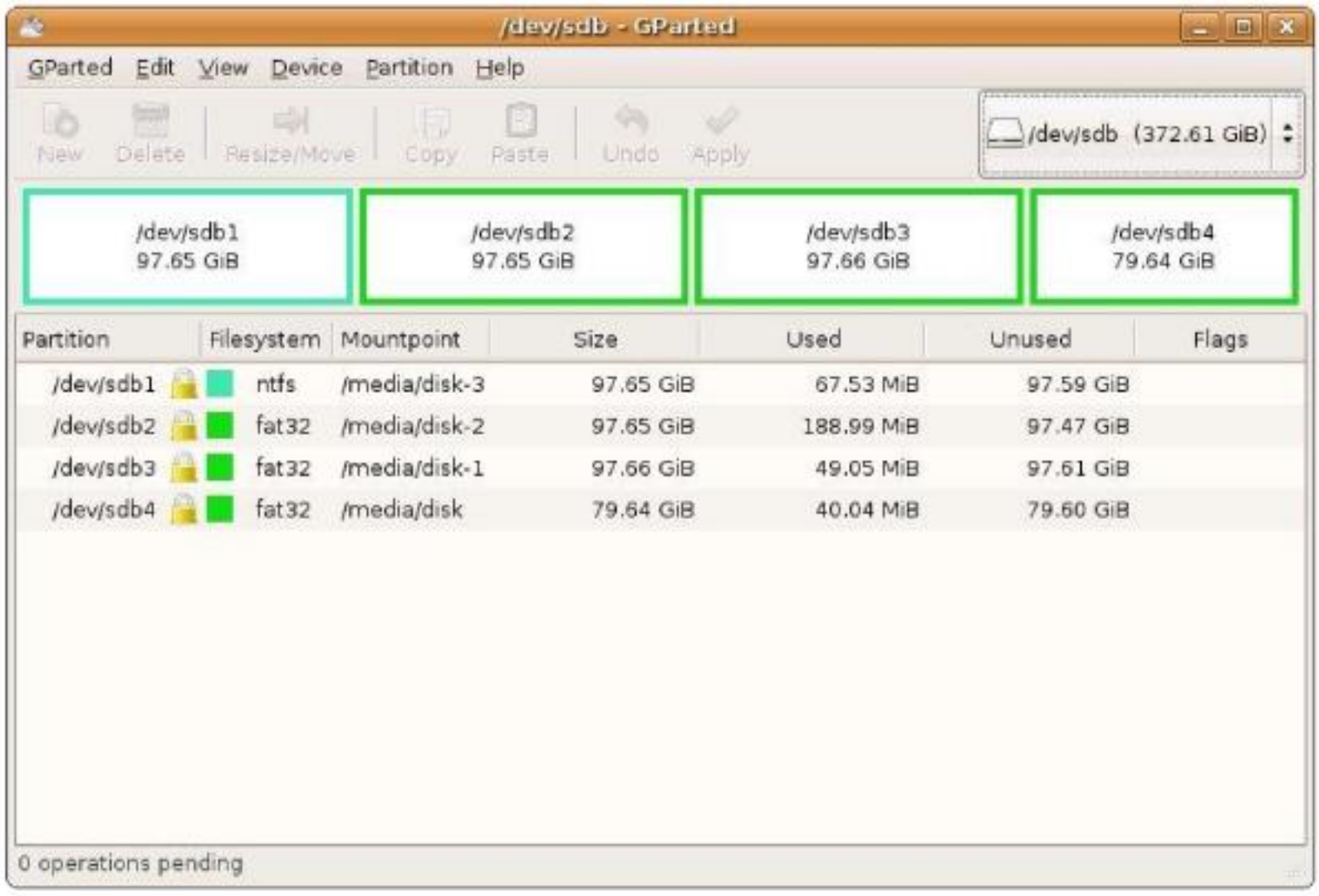


Figura 2.19: GParted

Întrebări

1. Câți utilizatori cu identificatorul 0 pot exista pe un sistem cu o partiție primară și două extinse:
   - ❑ niciunul
   - ❑ 1
   - ❑ 2
   - ❑ oricâți
2. Care dintre următoarele NU este necesară pentru a rula un sistem de operare de pe Live CD:
   - ❑ configurarea unei partiții primare
   - ❑ BIOS
   - ❑ RAM
   - ❑ procesor
3. Din ce sisteme de operare NU se pot realiza configurării din TUI?
   - ❑ Windows Vista
   - ❑ Debian
   - ❑ ubuntu Live CD
   - ❑ PalmOS

4. Operația de repornire a sistemului poate fi inițiată doar de utilizatorul privilegiat. În urma repornirii sistemului tot conținutul memoriei RAM este golit.
   - ❑ adevărat, adevărat
   - ❑ adevărat, fals
   - ❑ fals, adevărat
   - ❑ fals, fals
5. Care dintre variante este adevărată, dacă la execuția comenzii

```
1 su rrazvan
```

   se obține un prompt de shell fără a se solicita autentificare:
   - ❑ comanda a fost lansată de utilizatorul `root`
   - ❑ comanda a fost lansată de utilizatorul `rrazvan`
   - ❑ comanda a fost lansată din directorul `/etc`
   - ❑ comanda a fost lansată din directorul `/root`
6. Care din următoarele NU reprezintă un sistem de fișiere?
   - ❑ FAT32
   - ❑ swap
   - ❑ ext3
   - ❑ NTFS
7. Care din următoarele NU este un tip de partiție?
   - ❑ extinsă
   - ❑ logică
   - ❑ primară
   - ❑ virtuală
8. Care din următoarele distribuții este derivată din Debian?
   - ❑ Ubuntu
   - ❑ Fedora
   - ❑ Gentoo
   - ❑ OpenSuSE
9. Care din următoarele NU se referă la CLI?
   - ❑ comandă
   - ❑ prompt
   - ❑ terminal
   - ❑ window manager

10. Care din următoarele comenzi NU este o comandă de ajutor pe un sistem Linux?

- ❑ man
- ❑ info
- ❑ apropos
- ❑ adduser

Capitolul 3

Gestiunea pachetelor și utilizatorilor

Users */nm./: collective term for those who use computers. Users are divided into three types: novice, intermediate and expert.*
Novice Users*: people who are afraid that simply pressing a key might break their computer.*
Intermediate Users*: people who don't know how to fix their computer after they've just pressed a key that broke it.*
Expert Users*: people who break other people's computers.*

The Jargon File

Ce se învață din acest capitol?

- Gestiunea utilizatorilor și grupurilor în Linux
- Fișiere folosite pentru gestiunea utilizatorilor
- Ce este un pachet
- Care sunt operațiile de bază asupra unui pachet în Linux
- Gestiunea pachetelor în Debian/Ubuntu: utilitarele apt și dpkg
- Personal Package Archive (PPA)

3.1 Gestiunea utilizatorilor

Gestiunea utilizatorilor se referă la adăugarea de noi utilizatori, ștergerea unui utilizator existent, modificarea informațiilor despre un utilizator și afișarea diverselor informații.

În sistemele Unix, informațiile despre utilizatori sunt reținute în fișierul `/etc/passwd`. Fiecare linie din acest fișier conține numele utilizatorului, identificatorului său, home-ul, shell-ul rulat în momentul autentificării și alte informații de descriere:

```
root@anaconda:~# cat /etc/passwd
andreir:x:1114:1026:Andrei Rizoiu:/home/students/andreir:/bin/bash
alexn:x:1115:1026:Alex Negrea:/home/students/alexn:/bin/bash
[...]
```

Din motive de securitate, hash-ul asociat parolei nu se găsește în fișierul `/etc/passwd`, ci în fișierul `/etc/shadow` care nu poate fi accesat de majoritatea utilizatorilor (vezi secțiunea 10.2.3):

```
root@anaconda:~# ls -l  /etc/shadow
-rw-r----- 1 root shadow 7068 2008-09-12 11:59 /etc/shadow
```

Pentru a afla informații despre un utilizator al sistemului se pot folosi comenzile **`id`** sau **`finger`**:

```
root@anaconda:~# id andreir
uid=1114(andreir) gid=1026(students) groups=1026(students),1037(rl)

root@anaconda:~# finger alexn
Login: alexn                            Name: Alex Negrea
Directory: /home/students/alexn         Shell: /bin/bash
Never logged in.
No mail.
No Plan.
```

Utilizatorul privilegiat într-un sistem Unix este utilizatorul **root** cu uid-ul `0` și home-ul în `/root`:

```
root@anaconda:~# head -1 /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

Utilizatorul `root` (de fapt utilizatorul cu uid-ul 0) are drepturi absolute în cadrul sistemului și poate rula orice comandă. Se recomandă folosirea unui cont neprivilegiat. Doar atunci când este nevoie se va folosi contul privilegiat. Schimbarea unui utilizator se realizează cu ajutorul comenzii **`su`** urmată de introducerea parolei pentru acel utilizator. Dacă utilizatorul inițial este `root`, nu se solicită introducerea parolei:

```
root@anaconda:~# head -1 /etc/passwd
root:x:0:0:root:/root:/bin/bash

root@anaconda:~# su - andreir
andreir@anaconda:~$ su - razvan
Password:
```

Sistemele Ubuntu dezactivează, de obicei, utilizatorul `root` și recomandă folosirea comenzii **`sudo`**. De obicei, un utilizator neprivilegiat care are drept de sudo va rula comanda **`sudo bash`** pentru a obține un shell cu drepturi privilegiate (vezi secțiunea 10.3.3).

Schimbarea parolei unui utilizator se realizează cu ajutorul comenzii **`passwd`**. Utilizatorul privilegiat poate schimba parola oricărui utilizator. Un utilizator neprivilegiat își poate schimba parola doar sieși:

```
anaconda:~# passwd guest
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

3.1.1 UID, GID

Fiecărui utilizator din sistem îi este asociat un identificator numit `UID` (User Identifier). UID este un număr întreg ce poate lua valori între 0 și 32767 sau 65535, funcție de distribuție. Valoarea 0 este rezervată pentru utilizatorul `root`.

Pentru a găsi identitatea unui utilizator sistemul de operare folosește UID, și nu numele utilizatorului. De aceea, dacă în fișierul `/etc/passwd` se înlocuiește UID-ul unui utilizator normal cu 0, acel utilizator va căpăta drepturi administrative egale cu ale utilizatorului `root`.

Pe sistemele de operare Linux utilizatorii pot fi organizați în grupuri. În acest fel se pot realiza diverse configurări administrative mai ușor, aplicându-se politici la nivelul întregului grup și nu per utilizator. Un grup poate conține mai mulți utilizatori. Un utilizator poate face parte din mai multe grupuri.

Pentru fiecare grup sistemul de operare asociază un identificator, numit GID (Group Identifier). Acest identificator este un număr întreg ce poate lua valori între 0 și 32767 (limita superioară poate să difere în funcție de distribuție). GID 0 este rezervat pentru grupul utilizatorului root. La fel ca la UID, sistemul de operare folosește GID pentru a identifica un anumit grup și nu numele grupului.

3.1.2 Adăugarea și ștergerea utilizatorilor

Adăugarea unui utilizator poate fi realizată doar de `root`, în două moduri. Primul dintre ele este utilizarea comenzii **useradd**. Utilizarea acestei comenzi este greoaie, motiv pentru care adăugarea unui utilizator se realizează în mod uzual cu ajutorul comenzii **adduser**. **adduser** este un script care apelează la rândul lui **useradd**.

Apelarea **adduser** se poate realiza cu următorii parametri:

```
1 adduser [options] [--home DIR] [--shell|-s SHELL] [--no-create-home
  ] [--uid ID] [--firstuid ID] [--lastuid ID] [--ingroup GROUP | --gid ID]
    [--disabled-password] [--disabled-login] [--gecos GECOS] [--
  add_extra_groups] user
```

Dacă **adduser** este apelat doar cu numele utilizatorului ce va fi creat, el va porni un asistent care va cere toate datele necesare creării contului:

```
1  root@asgard:/home/george# adduser uso
2  Adding user 'uso' ...
3
4  Adding new group 'uso' (1001) ...
5
6  Adding new user 'uso' (1001) with group 'uso' ...
7
8  Creating home directory '/home/uso' ...
9
10 Copying files from '/etc/skel' ...
11
12 Enter new UNIX password:
13 Retype new UNIX password:
14 passwd: password updated successfully
15 Changing the user information for uso
16 Enter the new value, or press ENTER for the default
```

```
17          Full Name []: uso
18          Room Number []: EG306
19          Work Phone []:
20          Home Phone []:
21          Other []:
22  Is the information correct? [y/N] y
23
24  root@asgard:/home/george#
```

Din motive de securitate caracterele tastate în dreptul parolei nu vor fi afișate.

La crearea unui utilizator este creat și un grup cu același nume la care este adăugat utilizatorul creat.

Ştergerea unui utilizator se poate realiza cu **`userdel`** sau cu **`deluser`** (**`deluser`** este un script care apelează la rândul lui **`userdel`**):

```
1  root@asgard:/home/george# deluser uso
2  Removing user 'uso' ...
3
4  Done.
```

3.1.3 Adăugarea și ștergerea unui grup de utilizatori

Adăugarea unui grup de utilizatori se poate realiza cu ajutorul comenzii **`groupadd`**, sau cu ajutorul comenzii **`addgroup`**. **`addgroup`** este un script care apelează la rândul lui **`groupadd`**.

```
1  root@asgard:/home/george# addgroup usogroup
2  Adding group 'usogroup' (GID 1002) ...
3
4  Done.
```

Ştergerea unui grup de utilizatori se realizează tot prin două comenzi, **`groupdel`** și **`delgroup`** (**`delgroup`** este un script care apelează **`groupdel`**).

```
1  root@asgard:/home/george# delgroup usogroup
2  Removing group 'usogroup' ...
3
4  Done.
```

3.1.4 Modificarea datelor unui utilizator

Modificarea datelor unui utilizator se realizează cu ajutorul comenzii **`usermod`**. Aceasta permite modificarea grupului din care face parte un utilizator, a directorului de bază (*home*), a numelui de utilizator, a parolei etc.

De exemplu, pentru modifica directorul de bază al utilizatorului `george` se folosește opțiunea `-d`:

```
1  uso@asgard:~$ cd ~
2
3  uso@asgard:~$ pwd
4  /home/ uso
5
```

```
uso@asgard:~$ sudo su

root@asgard:/home/uso# usermod -d /home/usodir/ uso

root@asgard:/home/uso# exit

exit

uso@asgard:~$ exit

uso@asgard:~$ pwd
/home/uso

uso@asgard:~$
```

În exemplul anterior a fost schimbat directorul curent în directorul de bază al utilizatorului `uso`, apoi a fost afișat directorul curent (pentru a vedea care este directorul de bază actual), a fost pornit un nou shell în care utilizatorul autentificat a fost `root`, a fost modificat directorul de bază al utilizatorului `uso` în `/home/usodir`, a fost închis shell-ul de `root`, apoi și utilizatorul `uso` a părăsit sistemul de operare. Schimbarea directorului de bază al unui utilizator se realizează doar după ce utilizatorul părăsește sistemul și se autentifică din nou:

```
uso@asgard:~$ pwd
/home/usodir/
```

Modificarea parolei unui utilizator se poate face și cu ajutorul comenzii **`passwd`**:

```
root@asgard:/home/uso# passwd uso
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Din motive de securitate caracterele tastate în dreptul parolei nu vor fi afișate.

Dacă nu este specificat niciun nume de utilizator va fi schimbată parola utilizatorului curent. Doar utilizatorul `root` poate schimba parola altui utilizator.

3.1.5 Adăugarea și ștergerea utilizatorilor

În sistemele Debian-based, adăugarea, respectiv ștergerea unui utilizator se realizează prin intermediul scripturilor adduser și deluser:

```
root@anaconda:~# adduser test
Adding user 'test' ...
Adding new group 'test' (1038) ...
Adding new user 'test' (1003) with group 'test' ...
Creating home directory '/home/test' ...
Copying files from '/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for test
Enter the new value, or press ENTER for the default
        Full Name []: Test User
        Room Number []:
```

```
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [y/N] y

root@anaconda:~# id test
uid=1003(test) gid=1038(test) groups=1038(test)

root@anaconda:~# deluser --remove-home test
Looking for files to backup/remove ...
Removing files ...
Removing user `test' ...
Done.
```

Avantajul și, în același timp, dezavantajul folosirii comenzii **adduser** este interactivitatea. Automatizarea sarcinilor presupune comenzi non-interactive. Pentru aceasta, se pot folosi comenzile **useradd**, **userdel** și **usermod**. **useradd**, respectiv **userdel** sunt folosite de scripturile **adduser** și **deluser**.

```
root@anaconda:~# useradd -m -d /home/test test

root@anaconda:~# id test
uid=1116(test) gid=1116(test) groups=1116(test)

root@anaconda:~# usermod -s /bin/sh test

root@anaconda:~# userdel -r test

root@anaconda:~# id test
id: test: No such user
```

3.2 Gestiunea pachetelor

> Un **pachet**, sau un pachet software, este o aplicație sau o componentă accesibilă în forma unei arhive care poate fi instalată de un sistem de gestiune a pachetelor (**PMS** – *Package Management System*).

De obicei, pachetele reprezintă programe precompilate care pot fi instalate ușor, spre deosebire de instalarea din surse care este mai anevoioasă (vezi secțiunea 14.6.1).

Lucrul cu pachete (instalare, dezinstalare, configurare) se realizează prin intermediul unui sistem de gestiune a pachetelor (precum **dpkg**, **rpm**, **pacman**). Un astfel de sistem conține, în afară de fișierele asociate programului și un set de metainformații precum versiunea pachetului, descrierea și dependențele acestuia. PMS-ul folosește aceste informații pentru a decide dacă se va realiza instalarea pachetului, upgrade-ul acestuia, instalarea dependențelor etc. O dependență între pachetele A și B înseamnă că instalarea pachetului A necesită instalarea pachetului B. La fel, dezinstalarea pachetului B va forța dezinstalarea pachetului A.

Majoritatea distribuțiilor GNU/Linux folosesc noțiunea de *repository* (depozit de pachete). Acesta este un URL care precizează locația diverselor pachete ale distribuției. Aceste depozite sunt precizate în fișiere de configurare specifice distribuției.

Aplicații front-end peste PMS pot interoga depozitele și pot descărca și instala noi pachete.

În lumea Linux există diverse formate de pachete, cele mai cunoscute fiind formatul **DEB**, specific distribuțiilor Debian-based și formatul **RPM** folosit de Fedora/RedHat, Mandriva, SuSE, etc. Fiecare format are propriul PMS. Utilitarul **`alien`**[1] permite conversia între diverse formate de pachete.

Gestiunea pachetelor DEB

Utilitarul de bază (PMS) pentru gestiunea pachetelor DEB este **`dpkg`**. Folosit efectiv pentru instalarea sau dezinstalarea unui pachet, acest utilitar oferă opțiuni pentru interogarea stării actuale a pachetelor sau a conținutul acestora. Printre opțiunile utile se numără:

- listarea conținutului unui pachet

```
root@anaconda:/tmp# dpkg -L coreutils
/.
/bin
/bin/mkdir
/bin/mv
/bin/true
/bin/mknod
/bin/sleep
/bin/touch
/bin/chgrp
/bin/uname
/bin/echo
/bin/sync
[...]
```

- afișarea pachetelor al căror nume se potrivește cu o expresie regulată

```
root@anaconda:/tmp# dpkg -l 'linux*'
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-
installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err:
uppercase=bad)
||/ Name           Version        Description
+++-==============-==============-====================================

un  linux          <none>         (no description available)
un  linux-doc-2.6. <none>         (no description available)
un  linux-gnu      <none>         (no description available)
un  linux-image    <none>         (no description available)
un  linux-image-2. <none>         (no description available)
ii  linux-image-2. 2.6.18+6etch3  Linux kernel 2.6 image on Ppro/
Cele
[...]
```

- căutarea pachetului ce conține un anumit fișier

```
root@anaconda:/tmp# dpkg -S /bin/ps
procps: /bin/ps
```

[1] http://kitenet.net/~joey/code/alien/

În mod obișnuit, însă, cea mai mare parte a acestor acțiuni vor fi realizate prin intermediul utilitarului **APT** (*Advanced Packaging Tool*). APT este un front-end peste **dpkg** și permite interogarea depozitelor de pachete configurate, verificarea dependențelor, descărcarea automată a pachetelor din repository, actualizarea acestora, upgrade-ul unei distribuții etc.

Fișierul de configurare a unui depozit DEB pentru folosirea cu **APT** este `/etc/apt/sources.list`:

```
root@anaconda:/tmp# cat /etc/apt/sources.list
[...]
deb http://ftp.lug.ro/debian etch main contrib non-free
deb-src http://ftp.lug.ro/debian etch main contrib non-free
```

Adăugarea unui nou depozit înseamnă adăugarea unei noi linii în fișierul de configurare.

Acțiunile care pot fi realizate cu ajutorul utilitarului **apt** sunt:

- actualizarea listei de pachete

```
root@anaconda:/tmp# apt-get update
Get: 1 http://ftp.lug.ro etch Release.gpg [386B]
Get: 2 http://ftp.lug.ro etch/updates Release.gpg [189B]
Hit http://ftp.lug.ro etch Release
Get: 3 http://www.backports.org etch-backports Release.gpg [189B]
Get: 4 http://ftp.lug.ro etch/updates Release [37.6kB]
Ign http://debian.pkgs.cpan.org unstable Release.gpg
Get: 5 http://www.backports.org etch-backports Release [43.7kB]
Ign http://ftp.lug.ro etch/main Packages/DiffIndex
Ign http://ftp.lug.ro etch/contrib Packages/DiffIndex
[...]
```

- căutarea de pachete

```
root@anaconda:/tmp# apt-cache search hevea
hevea - translates from LaTeX to HTML, info, or text
lyx - High Level Word Processor
hevea-doc - HeVeA documentation
```

- afișarea de informații despre un fișier

```
root@anaconda:/tmp# apt-cache show hevea
Package: hevea
Priority: optional
Section: tex
Installed-Size: 2125
Maintainer: Debian OCaml Maintainers <debian-ocaml-maint@lists.
debian.org>
Architecture: all
Version: 1.09-3
```

- instalarea unui pachet (și a dependențelor sale)

```
root@anaconda:/tmp# apt-get install apt-file
Reading package lists... Done
Building dependency tree... Done
The following extra packages will be installed:
  libapt-pkg-perl libconfig-file-perl
The following NEW packages will be installed
  apt-file libapt-pkg-perl libconfig-file-perl
0 upgraded, 3 newly installed, 0 to remove and 66 not upgraded.
```

```
Need to get 106kB of archives.
After unpacking 406kB of additional disk space will be used.
Do you want to continue [Y/n]? Y
```

- dezinstalarea unui pachet

```
anaconda:/tmp# apt-get remove --purge apt-file
Reading package lists... Done
Building dependency tree... Done
The following packages will be REMOVED
  apt-file*
```

- curățarea cache-ului local de pachete

```
root@anaconda:/tmp# apt-get clean
instalarea surselor unui pachet

root@anaconda:/tmp# apt-get source apt-file
Reading package lists... Done
Building dependency tree... Done
Need to get 17.7kB of source archives.
Get: 1 http://ftp.lug.ro etch/main apt-file 2.0.8.2 (dsc) [505B]
Get: 2 http://ftp.lug.ro etch/main apt-file 2.0.8.2 (tar) [17.2kB]
```

- listarea tuturor pachetelor din cache-ul local de pachete, împreună cu dependețele acestora

```
root@anaconda:/tmp# apt-cache dump
Using Versioning System: Standard .deb
Package: pipenightdreams
 Version: 0.10.0-13
     File: /var/lib/apt/lists/ro.archive.ubuntu.
com_ubuntu_dists_jaunty_universe_binary-i386_Packages
  Depends: libc6 2.6.1-1
  Depends: libgcc1 1:4.2.1
  Depends: libsdl-image1.2 1.2.5
  Depends: libsdl1.2debian 1.2.10-1
  Depends: libstdc++6 4.2.1
  Depends: pipenightdreams-data 0.10.0-13
 Description Language:
                 File: /var/lib/apt/lists/ro.archive.ubuntu.
com_ubuntu_dists_jaunty_universe_binary-i386_Packages
                  MD5: 7d042c60ae2f422df1ada8160fb80333
...
```

- listarea repository-urilor din care face parte un pachet, versiunea disponibilă, versiunile candidate etc.

```
root@anaconda:/tmp#  apt-cache policy firefox
firefox:
  Installed: 3.0.14+build2+nobinonly-0ubuntu0.9.04.1
  Candidate: 3.0.14+build2+nobinonly-0ubuntu0.9.04.1
  Version table:
 *** 3.0.14+build2+nobinonly-0ubuntu0.9.04.1 0
        500 http://ro.archive.ubuntu.com jaunty-updates/main
Packages
        500 http://security.ubuntu.com jaunty-security/main Packages
        100 /var/lib/dpkg/status
     3.0.8+nobinonly-0ubuntu3 0
        500 http://ro.archive.ubuntu.com jaunty/main Packages
```

PPA

Există și conceptul de **Personal Package Archives**, un sistem ce-i permite programatorului să-și încarce pachetele proprii pentru a fi incluse în update-urile ulterioare de pachete, fiind astfel mai ușor distribuite către utilizatorii finali. Astfel, acesta va putea instala programul scris de altcineva ca pe un pachet normal.

Totuși, pentru a preveni problemele de securitate, acest lucru nu poate fi făcut oarecum. În continuare vom enumera pașii necesari pentru a putea instala programe dintr-un PPA, numit *someone561* necesar pentru a instala o versiune mai recentă a unor compilatoare.

1. Vom crea un fișier nou cu un conținut și o locație ca mai jos pentru a oferi detalii cu privire la adresa de unde se vor descărca pachetele

```
1 mihai@keldon:/tmp# cat /etc/apt/sources.list.d/haskell.list
2 deb http://ppa.launchpad.net/someone561/ppa/ubuntu jaunty main
3 deb-src http://ppa.launchpad.net/someone561/ppa/ubuntu jaunty main
```

   (se va schimba jaunty cu numele versiunii distribuției folosite)

2. Pasul următor constă în autentificarea acestui PPA. Deși poate fi făcută și prin editarea unui fișier sau este opțională, acest lucru nu este recomandat. Se preferă folosirea următoarei comenzi:

```
1 mihai@keldon:/tmp# apt-key adv --recv-keys --keyserver keyserver.
  ubuntu.com E51D9310
```

   unde ultimul argument reprezintă un cod unic de autentificare al posesorului PPA-ului. Dacă nu dorim să realizăm autentificarea vom avea mereu mesaje de avertisment în cazul instalării unor pachete din aceste repository-uri.

3. Ultimul pas va consta în reactualizarea cache-ului local de pachete și instalarea pachetelor din acest PPA

```
1 mihai@keldon:/tmp# apt-get update
2 ...
3 mihai@keldon:/tmp# apt-get install ghc6 ...
4 ...
```

update-alternatives

În timpul rulării Linux, este un caz obișnuit să existe mai multe programe având aproximativ același scop. În special, compilatoarele și editoarele de text sunt exemple des întâlnite ale acestui caz. Prin urmare, deseori apare problema alegerii programului potrivit.

Această problemă este accentuată de faptul că există multe programe ce vor apela alte programe după un nume – de exemplu, multe programe permit utilizatorilor să editeze un fișier, fără a fi ele însele un editor. Ar trebui deci rulat un alt editor din sistem, dar care din ele? Un exemplu ar fi managementul taskurilor utilizând cron.

Prin urmare, pentru o clasă de aplicații care pot folosi o suită de alte programe, este nevoie de o modalitate de a configura variante alternative. O variantă simplă ar fi folosirea unor variabile de mediu (EDITOR de exemplu).

Varianta propusă de Debian este de a avea un set de comenzi standard, precum **`editor`**, **`www-browser`**, **`view`** etc. De fapt, toate acestea sunt link-uri simbolice spre comanda reală.

Pentru a seta aceste link-uri, se va folosi comanda **`update-alternatives`**. Acest sistem poate funcționa în două moduri:

- automatic mode – sistemul decide singur către ce program vor indica link-urile
- manual mode – administratorul sistemului va decide acest lucru

Implicit, modul este automatic. Link-urile simbolice sunt structurate în grupuri conținând prioritățile diverselor programe (editorul cu prioritate maximă este cel ales în cazul rulării **`editor`**).

Putem realiza următoarele:

- listarea informațiilor despre un grup

```
mihai@keldon:~# update-alternatives --display editor
editor - status is auto.
 link currently points to /usr/bin/vim.gnome
/usr/bin/vim.tiny - priority 10
 slave editor.ru.1.gz: /usr/share/man/ru/man1/vim.1.gz
 slave editor.pl.ISO8859-2.1.gz: /usr/share/man/pl.ISO8859-2/man1/
vim.1.gz
 slave editor.it.ISO8859-1.1.gz: /usr/share/man/it.ISO8859-1/man1/
vim.1.gz
 slave editor.1.gz: /usr/share/man/man1/vim.1.gz
 slave editor.pl.UTF-8.1.gz: /usr/share/man/pl.UTF-8/man1/vim.1.gz
 slave editor.it.1.gz: /usr/share/man/it/man1/vim.1.gz
 slave editor.fr.UTF-8.1.gz: /usr/share/man/fr.UTF-8/man1/vim.1.gz
 slave editor.fr.1.gz: /usr/share/man/fr/man1/vim.1.gz
 slave editor.it.UTF-8.1.gz: /usr/share/man/it.UTF-8/man1/vim.1.gz
 slave editor.pl.1.gz: /usr/share/man/pl/man1/vim.1.gz
 slave editor.fr.ISO8859-1.1.gz: /usr/share/man/fr.ISO8859-1/man1/
vim.1.gz
/bin/ed - priority -100
 slave editor.1.gz: /usr/share/man/man1/ed.1.gz
/bin/nano - priority 40
 slave editor.1.gz: /usr/share/man/man1/nano.1.gz
/usr/bin/vim.gnome - priority 60
 slave editor.ru.1.gz: /usr/share/man/ru/man1/vim.1.gz
 slave editor.pl.ISO8859-2.1.gz: /usr/share/man/pl.ISO8859-2/man1/
vim.1.gz
 slave editor.it.ISO8859-1.1.gz: /usr/share/man/it.ISO8859-1/man1/
vim.1.gz
 slave editor.1.gz: /usr/share/man/man1/vim.1.gz
 slave editor.pl.UTF-8.1.gz: /usr/share/man/pl.UTF-8/man1/vim.1.gz
 slave editor.it.1.gz: /usr/share/man/it/man1/vim.1.gz
 slave editor.fr.UTF-8.1.gz: /usr/share/man/fr.UTF-8/man1/vim.1.gz
 slave editor.fr.1.gz: /usr/share/man/fr/man1/vim.1.gz
 slave editor.it.UTF-8.1.gz: /usr/share/man/it.UTF-8/man1/vim.1.gz
 slave editor.pl.1.gz: /usr/share/man/pl/man1/vim.1.gz
 slave editor.fr.ISO8859-1.1.gz: /usr/share/man/fr.ISO8859-1/man1/
vim.1.gz
Current 'best' version is /usr/bin/vim.gnome.
```

- listarea tuturor țintelor din grup

```
mihai@keldon:~# update-alternatives --list editor
/usr/bin/vim.tiny
/bin/ed
/bin/nano
/usr/bin/vim.gnome
```

- schimbarea interactivă a unei opțiuni

```
root@keldon:/home/mihai# update-alternatives --config www-browser

There are 2 alternatives which provide 'www-browser'.

  Selection    Alternative
-----------------------------------------------
          1    /usr/bin/w3m
*+        2    /usr/bin/elinks

Press enter to keep the default[*], or type selection number:
```

- schimbarea unui link simbolic (neinteractiv)

```
root@keldon:/home/mihai# update-alternatives --set www-browser /usr/
bin/elinks
Using '/usr/bin/elinks' to provide 'www-browser'.
```

Pentru configurarea tuturor alternativelor se folosește **`update-alternatives --all`**

3.3 Studii de caz

3.3.1 Fișierele în care sunt stocate informații despre utilizatori

Informațiile despre grupuri și utilizatori sunt stocate în trei fișiere text:

- `/etc/passwd` – informații despre conturile utilizatorilor
- `/etc/shadow` – informații despre parolele utilizatorilor
- `/etc/group` – informații despre grupurile de utilizatori

Fișierul `/etc/passwd` conține câte o linie pentru fiecare utilizator. Această linie are șapte câmpuri despărțite prin caracterul ":". Câmpurile sunt (în ordinea în care apar în fișier):

- Nume de utilizator
- Parola. Un caracter x în această poziție arată faptul că parola este stocată în fișierul /etc/shadow.
- UID
- GID
- Informații legate de utilizator (Nume, Adresa, Telefon)
- Directorul de bază (directorul *home*)

- *Shell*-ul folosit

Un exemplu de fișier `/etc/passwd` este:

```
root@asgard:/home/george# cat /etc/passwd
[...]
sync:x:4:65534:sync:/bin:/bin/sync
george:x:1000:1000:George,,,:/home/george:/bin/bash
uso:x:1001:1001:uso,EG306,,:/home/uso:/bin/bash
[...]
```

Inițial, tot în fișierul `/etc/passwd` se rețineau și parolele, dar, pe măsură ce atacurile bazate pe dicționare au devenit fezabile, parolele au fost mutate într-un fișier cu permisiuni `600` numit `/etc/shadow`.

Fișierul `/etc/shadow` conține și el câte o linie pentru fiecare utilizator. Pe fiecare linie se găsesc opt câmpuri despărțite prin caracterul ":". Câmpurile sunt (în ordinea în care apar în fișier):

- Numele de utilizator
- Parola în format criptat. Dacă în loc de parolă se află un caracter "!", utilizatorul respectiv nu se poate autentifica în sistem
- Data la care a fost realizată ultima schimbare de parolă (data este notată în numărul de zile trecute de la 1 ianuarie 1970)
- Numărul minim de zile care pot trece între două schimbări de parolă
- Numărul maxim de zile pentru care parola este activă
- Cu câte zile înainte de expirarea parolei este anunțat utilizatorul să își shimbe parola
- Numărul de zile care trec după expirarea parolei până când contul va fi dezactivat
- Data la care contul va expira (data este notată în numărul de zile trecute de la 1 ianuarie 1970)

Criptarea parolei se face folosind o funcție de hash, în general MD5[1], dar, de preferat, SHA-2[2]. Aceste funcții sunt *one-way*, deci nu se poate afla parola simplu nici măcar folosind `/etc/shadow`. La fiecare autentificare, parola introdusă de utilizator este trecută prin algoritmul de hash și comparată cu valoarea din `/etc/shadow`.

Mai multe detalii despre criptarea parolei se poat afla din pagina de manual a comenzii **`crypt`**.

Un exemplu de fișier `/etc/shadow` este:

```
root@asgard:/home/georgedir/# cat /etc/shadow
root:!:13774:0:99999:7:::
daemon:*:13620:0:99999:7:::
bin:*:13620:0:99999:7:::
sys:*:13620:0:99999:7:::
sync:*:13620:0:99999:7:::
george:$1$LBNk03w4$4O2R/uMfCs/R9LOmH6zDG0:13774:0:99999:7:::
uso:$1$IPSj/ETo$1Uqe1Nrqm0r4LYdMihh6x1:13775:0:99999:7:::
```

[1] http://en.wikipedia.org/wiki/MD5
[2] http://en.wikipedia.org/wiki/SHA-2

```
9 [...]
```

Fișierul `/etc/group` conține câte o linie pentru fiecare grup. Această linie are patru câmpuri despărțite prin caracterul ":". Câmpurile sunt (în ordinea în care apar în fișier):

- Numele grupului
- Parola (câmp ce nu mai este folosit)
- GID
- Numele utilizatorilor ce fac parte din grup, în afară de utilizatorul care are același nume cu numele grupului

Un exemplu de fișier `/etc/group` este:

```
1 root@asgard:/home/georgedir/# cat /etc/group
2 root:x:0:
3 lp:x:7:cupsys
4 scanner:x:104:cupsys,hplip,george
5 george:x:1000:
6 uso:x:1001:
7 [...]
```

3.3.2 Actualizarea unui sistem Debian sau Ubuntu

Utilitarul **`apt`** oferă posibilitatea de a actualiza toate pachetele instalate pe un sistem care folosește pachete de tip `.deb`.

Primul pas în acest sens este asigurarea că lista locală de pachete este actualizată. Acest lucru se face folosind **`apt-get update`**. După această operație, utilizatorul are două alternative pentru instalarea celor mai noi pachete disponibile:

- **`apt-get upgrade`** actualizează toate pachetele care pot fi actualizate *fără a instala sau dezinstala alte pachete*. Este, din acest motiv, alternativa mai conservativă, pentru că nu introduce dependențe adiționale, și nici nu dezinstalează programe potențial utilizate;
- **`apt-get dist-upgrade`** este mai agresiv și instalează sau dezinstalează pachete după cum dependențele versiunilor noi cer. De asemenea, această comandă soluționează eventualele conflicte[1] într-un mod "inteligent", adică, în cazul unui conflict, va actualiza pachetul pe care îl consideră mai important.

În cele din urmă, o comandă utilă după un upgrade este **`apt-get autoremove`**. Aceasta va dezinstala pachetele care nu au fost instalate explicit de utilizator, dar nici nu mai sunt necesare niciunui pachet actual.

Cuvinte cheie

- utilizator
- uid
- grup
- gid

[1] Un conflict apare când două pachete cer explicit versiuni diferite dintr-un al treilea, ori când unul cere instalarea, iar celălalt dezinstalarea.

- root
- id
- finger
- /etc/passwd
- /etc/shadow
- /etc/group
- passwd
- su
- adduser, deluser
- useradd, userdel
- pachet
- Package Mangement System
- DEB, RPM
- /etc/apt/sources.list
- depozit (*repository*)
- apt
- dpkg
- dependență
- instalare, dezinstalare
- actualizare (*update*)
- upgrade

Întrebări

1. Care este comanda folosită pentru adăugarea non-interactivă a unui utilizator în sistem?
   - ❑ **`adduser`**
   - ❑ **`useradd`**
   - ❑ **`newuser`**
   - ❑ **`finger`**
2. De ce nu sunt ținute hash-urile parolelor în fișierul `/etc/passwd`?
   - ❑ `/etc/passwd` se poate afla pe o partiție nesigură.
   - ❑ permisiunile asupra `/etc/passwd` nu asigură securitate maximă.
   - ❑ întrebarea este o capcană, hash-urile parolelor chiar se află în `/etc/passwd`.
   - ❑ `/etc/passwd` nu poate fi modificat cu ajutorul comenzii **`passwd`**, de aceea este nevoie de un fișier cu permisiuni speciale.
3. Care dintre următoarele afirmații este adevărată despre sistemele de pachete?
   - ❑ `rpm` este bazat pe `apt`
   - ❑ `dpkg` este un wrapper peste `apt`
   - ❑ `apt` este un wrapper peste `dpkg`
   - ❑ sistemele Red Hat folosesc `dpkg`
4. Care este mecanismul prin care **`update-alternatives`** schimbă programele folosite pentru anumite task-uri?
   - ❑ legături simbolice

- ❑ legături hard
- ❑ instalare și dezinstalare on-demand a pachetelor
- ❑ script-uri wrapper care citesc configurația de fiecare dată când programul este rulat și lansează alternativa corespunzătoare

5. Care dintre următoarele informații NU este reținută în `/etc/passwd`?
   - ❑ directorul *home* al utilizatorului
   - ❑ interpretorul de comenzi (*shell*-ul)
   - ❑ grupurile din care face parte
   - ❑ numele utilizatorului
6. Un Personal Package Archive (PPA):
   - ❑ conține versiuni ale unor pachete diferite de ceea se găsește în repository-urile centrale
   - ❑ poate înlocui în întregime repository-urile centrale
   - ❑ este inerent nesigur și nu poate fi protejat
   - ❑ poate fi folosit de orice distribuție

Capitolul 4

Sisteme de fișiere

The Unix 'file system'. Sure it corrupts your data, but look how fast it is!

Ce se învață din acest capitol?

- Ce sunt și cum se utilizează sistemele de fișiere
- Tipuri de sisteme de fișiere
- Utilizarea căilor relative și absolute
- Lucrul cu fișiere (creare, copiere, mutare etc.)
- Descriptori de fișiere
- Redirectarea comenzilor în fișiere
- Arhivarea și dezarhivarea
- Drepturi de acces la fișiere

Cititorii familiarizați cu noțiunile de fișier, sisteme de fișiere și cu terminologia asociată pot porni direct de la secțiunea 4.3, continuând apoi cu analiza detaliată a sistemelor de fișiere.

4.1 Noțiuni introductive

4.1.1 Ce este un sistem de fișiere

Fișierul (*file*) este reprezentarea logică a unei informații sub forma unei înșiruiri de octeți. Fișierul poate fi considerat ca fiind versiunea electronică a documentului scris.

Directorul (*directory*) este o entitate în care se pot regăsi fișiere și/sau alte directoare. Acesta poate fi considerat versiunea electronică a dosarului.

În interfețele grafice, directorul este de obicei denumit folder.

Fișierele organizează informațiile pe mediile de stocare. Mediile de stocare pot fi considerate spații continue de octeți. În aceste spații, se pot regăsi mai multe fișiere, de dimensiuni variabile.

Sistemul de fișiere reprezintă modul de organizare a fișierelor pe un mediu de stocare pentru a le face mai ușor accesibile. Organizarea include atât partea logică (modul în care sunt adresate fișierele) cât și partea fizică (modul în care sunt stocate fișierele ca înșiruire de octeți).

Fiind o componentă a sistemului de operare, sistemul de fișiere menține numele și atributele fișierelor și permite stocarea lor într-o ierarhie de directoare numită și arbore de directoare.

Pentru ca utilizatorii să poată avea acces la fișiere, sistemul de operare oferă o interfață pentru lucrul cu sistemul de fișiere. Această interfață poate fi una textuală în mod text cum este interpretorul de comenzi, sau o interfață grafică.

Orice mediu de stocare poate fi utilizat pentru scrierea și citirea de fișiere dacă se cunoaște sistemul de fișiere utilizat pentru organizare.

Sistemele de operare folosesc fișierele pentru a organiza date, indiferent dacă aceste date sunt ale utilizatorului sau sunt generate pe moment de sistem.

Spre exemplu, în Linux și în Mac OS X, orice informație se găsește într-un fișier: datele se regăsesc în fișiere; directoarele sunt și ele fișiere, dar cu atributul de director; fiecare dispozitiv poate fi accesat printr-un fișier, inclusiv mouse-ul, memoria și placa video.

Intern, Windows folosește o schemă asemănătoare structurii de fișiere și directoare pentru a denumi dispozitivele, dar această structură nu se suprapune peste structura de fișiere precum în Linux și în Mac OS.

4.1.2 Ierarhia sistemului de fișiere

Sistemele de fișiere permit utilizatorului să organizeze datele într-un mod accesibil. Structura cel mai des întâlnită pentru organizarea fișierelor este arborele. Tabelele de mai jos prezintă structura ierarhică din sistemele de operare cele mai cunoscute.

În Tabela 4.1 se găsește o reprezentare grafică parțială pentru o ierarhie dintr-un sistem de fișiere în Linux. În rădăcină (`/`) se regăsesc directoarele `home`, `bin`, `usr` ..., în `home` se regăsesc directoarele `ubuntu` și `myuser` ș.a.m.d.

Comparativ cu Linux, structura în Windows este mult mai simplă pentru directoarele aflate imediat în rădăcină. În schimb, o mare parte din directoarele importante se ascund în directorul `Windows`.

Tabelul 4.1: Ierarhia într-un sistem de fișiere din mediul Unix

Director	Conținut
/	directorul rădăcină
/bin	comenzi esențiale necesare boot-ării, întreținerii și depanării sistemului
/boot	fișiere necesare boot-ării, precum imaginea kernel-ului
/dev	fișiere speciale utilizate pentru accesul direct la dispozitivele hardware sau logice ale sistemului
/etc	fișiere pentru configurarea sistemului, precum inittab, fstab și hosts
/home	fișierele fiecărui utilizator din sistem – datele unui utilizator se găsesc în /home/username
/media	subdirectoare în care se montează unitățile optice, floppy etc.
/mnt	subdirectoare în care se montează alte sisteme de fișiere
/opt	pachete de aplicații de dimensiuni mari, accesibile tuturor utilizatorilor
/proc	sistem virtual de fișiere din care se obțin informații despre sistem și aplicațiile care rulează la un moment dat
/root	directorul home al utilizatorului root
/sbin	comenzi de bază accesibile numai utilizatorului root
/tmp	fișiere temporare
/usr	aplicații pentru uzul normal al sistemului de operare – /usr/local conține aplicațiile instalate/compilate de utilizator
/var	fișiere al căror conținut se schimbă foarte des, precum log-uri, fișiere temporare, cache (date reutilizabile), spool (date neprocesate)

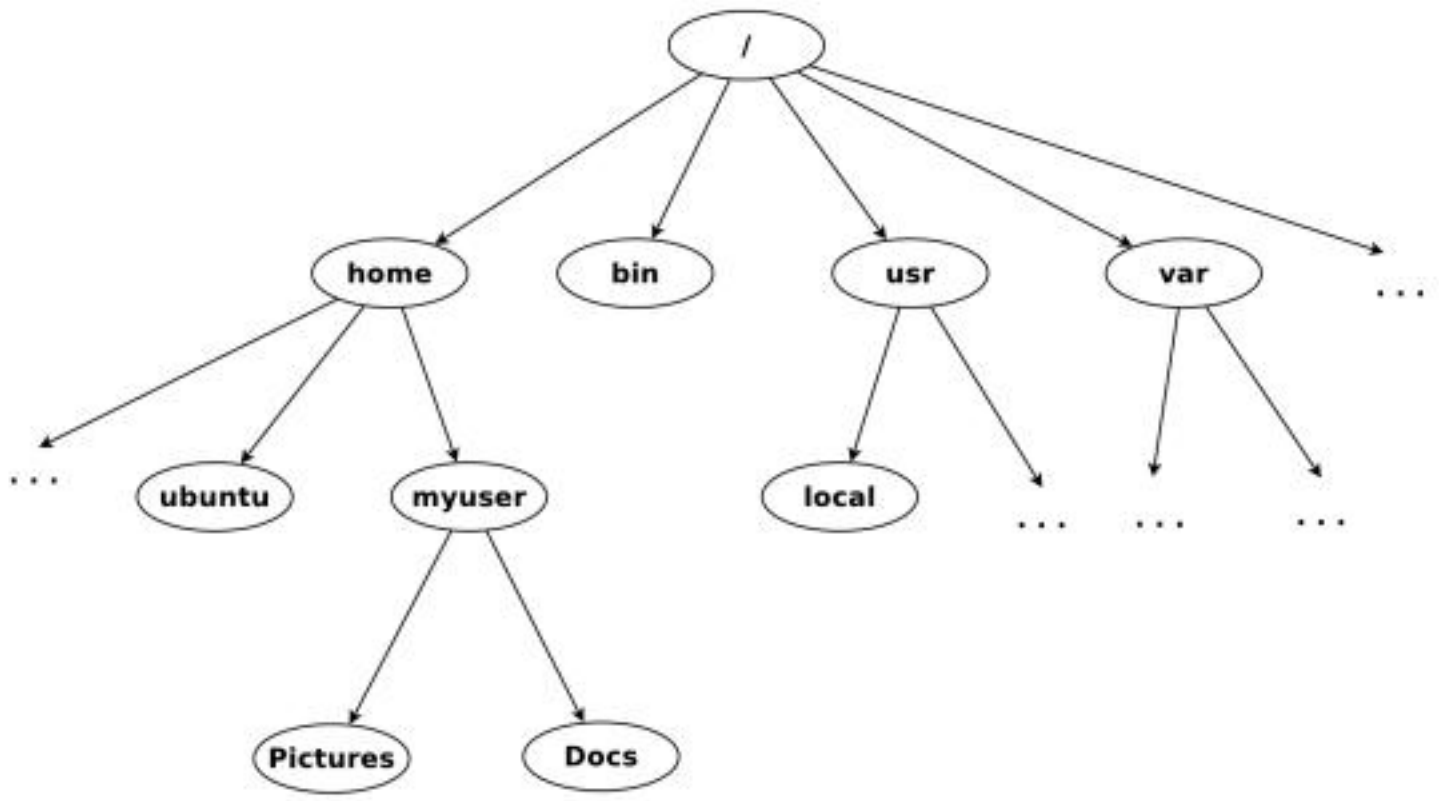


Figura 4.1: Ierarhia sistemului de fișiere în Linux

Tabelul 4.2: Ierarhia într-un sistem de fișiere din mediul Windows XP

Director	**Conținut**
`C:\`	directorul rădăcină
`C:\Windows`	Windows-ul și fișierele aferente
`C:\Documents and Settings`	configurările utilizatorilor și date specifice acestora
`C:\Program Files`	Aplicații
`C:\Windows\System32`	drivere și fișiere de configurare Windows
`C:\Documents and Settings\username\My Documents`	datele unui utilizator (aceasta este calea implicită, ea poate fi modificată)

Sistemul de fișiere folosit de Windows are o particularitate: există mai multe directoare rădăcină, câte unul pentru fiecare partiție sau disc. Spre exemplu, pot exista simultan următoarele directoare rădăcină:

- `A:` – floppy disk (de obicei literele `A:` și `B:` sunt rezervate de Windows pentru floppy disk-uri);
- `C:` – partiție de pe hard disk; dacă există mai multe partiții se asociază litere în ordine pentru fiecare dintre ele;
- `D:` – CD-ROM/DVD-ROM (următoarea literă disponibilă după ce s-au asociat litere partițiilor de pe hard disk(uri)).

După cum s-a precizat anterior, pe o partiție poate exista un singur sistem de fișiere la un moment dat. De obicei, sistemele de fișiere cu suport fizic se găsesc pe o singură partiție. Trebuie menționat faptul că Windows alocă literele după partiții, nu după sistemul de fișiere. Astfel, dacă sistemul de fișiere de pe o partiție se schimbă, acesta va avea asociată tot litera `C:`.

Tabelul 4.3 prezintă o analogie între căile importante din sistemele de operare majore existente în prezent.

4.1.3 Căi relative și căi absolute

Există două moduri prin care se poate accesa un fișier sau un director: folosind o cale absolută sau o cale relativă.

Calea absolută este calea care începe cu directorul rădăcină al sistemului de fișiere – în cazul Linux/Mac OS X este `/`; în cazul Windows este `C:`, `D:` etc.

Calea relativă este acea cale care indică spre un fișier pornind de la directorul curent.

Fiecare director conține două directoare speciale:

- `.` (punct) indică spre același director (directorul curent);
- `..` (punct, punct) indică spre directorul părinte.

Tabelul 4.3: Comparație între căile sistemelor de operare

Descriere	Windows	Linux	Mac OS X
radacină	`C:`	`/`	`/`
director home	`C:\Documents and Settings\username` (WinXP), `C:\Users\username` (Vista)	`/home/username`	`/Users/username`
aplicații	`C:\Program Files`	`/bin`; `/sbin`; `/usr/bin`; `/usr/sbin`; `/usr/local/bin` `/opt/*/bin`	`/Applications`; `/bin`; `/sbin`
configurări ale sistemului	Windows Registry	directoare specifice fiecărei aplicații, aflate în home-ul utilizatorului; `/etc`	`/Users/username/Library`; `/etc`

Astfel, dacă se dorește "urcarea" în ierarhia de fișiere, se folosește `..` care indică directorul părinte. Se poate urca mai mult în ierarhia de directoare înlănțuind câteva grupări de `..` delimitate prin separatorul de directoare.

În exemplul de mai jos:

- comanda **`pwd`** (*print working directory*) afișează directorul curent;
- comanda **`cd ../../..`** schimbă directorul curent în părintele-părintelui-părintelui directorului curent (s-ar putea spune "străbunicul directorului curent").

```
1 ubuntu@ubuntu:~/cs/uso$ pwd
2 /home/ubuntu/cs/uso
3
4 ubuntu@ubuntu:~/cs/uso$ cd ../../..
5
6 ubuntu@ubuntu:/home$ pwd
7 /home
```

`.` se folosește pentru a indica în mod explicit o cale care are ca punct de pornire directorul curent. `.` este utilizat frecvent pentru a scrie comenzi care execută script-uri/programe aflate în directorul curent.

Spre exemplu, comanda de mai jos execută fișierul `program_din_directorul_curent` – acest fișier trebuie să poată fi executat de utilizatorul curent.

```
1 ubuntu@ubuntu:~$ ./program_din_directorul_curent
```

Pentru mai multe exemple de utilizare a căilor relative și absolute se poate consulta secțiunea 4.4.

4.1.4 Variabila de mediu PATH

În mod normal, pentru execuția programelor uzuale, nu este necesară specificarea căii complete sau schimbarea directorului. Sistemele de operare utilizează variabile de mediu (environment variables) pentru a stoca diferite informații. `PATH` este o variabila de mediu care conține lista căilor uzuale unde se găsesc programe. Programele care se găsesc în oricare din căile din `PATH` pot fi executate fără a preciza calea către ele. Mai multe informații despre variabile de mediu și modificarea acestora se regăsesc în secțiunea 12.10.1.

Pentru a afișa variabila de mediu `PATH` într-un mediu Linux/Mac OS X se utilizează comanda:

```
1 ubuntu@ubuntu:~$ echo $PATH
2 /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

Căile din listă sunt despărțite prin separatorul `:` (două puncte).

Într-un mediu Windows, comanda este:

```
1 C:> echo %PATH%
2 C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem
```

Separatorul folosit în Windows este `;` (punct și virgulă).

4.2 Tipuri de fișiere

4.2.1 Terminologie

Sistemele de fișiere pun la dispoziție utilizatorilor diferite forme logice de organizare și acces la date, bineînțeles sub formă de fișiere.

În afară de directoare și fișiere, sistemul de fișiere pune la dispoziția utilizatorului și legături (link-uri).

Astfel, sistemele de fișiere moderne oferă posibilitatea utilizării mai multor tipuri de fișiere, prezentate în tabelul 4.4.

Tabelul 4.4: Tipuri de fișiere

Tipuri fișier	Denumire tip în engleză	Descriere
fișiere normale	regular files	unitate logică de acces la date
directoare	directories/folders	organizează fișiere și alte directoare
legături simbolice	symbolic links	legătura către un alt fișier

Există posibilitatea de a crea legături atât pentru fișiere cât și pentru directoare – în fond, ambele entități sunt fișiere (dar cu atribute diferite). Legăturile se utilizează la

scriere/citire la fel fișierele normale. Sistemul de operare împreună cu sistemul de fișiere se ocupă transparent de "traducerea" acestora în fișiere normale pentru aplicații.

Link-urile sunt create cu ajutorul comenzii **`ln`**. Mai jos sunt prezentate două exemple de folosire. În primul exemplu se creează un link al fișierului `/etc/apt/sources.list`, iar apoi la un fișier local precizându-se un nou nume pentru link.

```
ubuntu@ubuntu:~/tmp$ ls -l
total 1
-rw-r--r-- 1 ubuntu ubuntu  37 2007-09-04 10:59 hello.txt

ubuntu@ubuntu:~/tmp$ ln -s /etc/apt/sources.list

ubuntu@ubuntu:~/tmp$ ls -l
total 2
-rw-r--r-- 1 ubuntu ubuntu  37 2007-09-04 10:59 hello.txt
lrwxrwxrwx 1 ubuntu ubuntu  17 2007-09-04 10:59 sources.list -> /etc/apt/
sources.list

ubuntu@ubuntu:~/tmp$ ln -s hello.txt hello_link.txt

ubuntu@ubuntu:~/tmp$ ls -l
total 3
lrwxrwxrwx 1 ubuntu ubuntu   9 2007-09-04 11:00 hello_link.txt -> hello.
txt
-rw-r--r-- 1 ubuntu ubuntu  37 2007-09-04 10:59 hello.txt
lrwxrwxrwx 1 ubuntu ubuntu  17 2007-09-04 10:59 sources.list -> /etc/apt/
sources.list
```

4.2.2 Detecția tipului fișierelor

Pe sistemele Windows, detecția tipului fișierelor se bazează pe verificarea extensiei. În funcție de aceasta, sistemul de operare determină care aplicație este potrivită pentru a deschide un fișier.

Pe sistemele Linux, detecția tipului fișierelor este realizată prin intermediul comenzii **`file`**. Aceasta utilizează un fișier special de configurare în care sunt înscrise secvențe de octeți care se regăsesc în diferite tipuri de fișiere. **`file`** lucrează independent de extensia fișierului, precum este dovedit în exemplele următoare:

```
ubuntu@ubuntu:~$ file photo.jpg
photo.jpg: JPEG image data, JFIF standard 1.01

ubuntu@ubuntu:~$ mv photo.jpg fisier.txt

ubuntu@ubuntu:~$ file fisier.txt
photo.txt: JPEG image data, JFIF standard 1.01
```

4.3 Operații uzuale asupra fișierelor și directoarelor

4.3.1 Afișarea și schimbarea directorului curent

Comanda **pwd** afișează calea absolută către directorul curent. De asemenea, prompt-ul bash conține implicit directorul curent. Pentru a schimba directorul curent se folosește comanda **cd <cale>**. Calea poate să fie o cale absolută sau relativă.

Mai jos se găsește un exemplu de folosire a comenzilor **pwd** și **cd** – a se observa că la fiecare schimbare de director se modifică și prompt-ul bash. Ierarhia de directoare prin care se navighează este cea prezentată anterior.

```
ubuntu@ubuntu:~$ pwd
/home/ubuntu

ubuntu@ubuntu:~$ cd ..

ubuntu@ubuntu:/home$ pwd
/home

ubuntu@ubuntu:/home$ cd ../usr/bin

ubuntu@ubuntu:/usr/bin$ pwd
/usr/bin

ubuntu@ubuntu:/usr/bin$ cd .

ubuntu@ubuntu:/usr/bin$ cd /

ubuntu@ubuntu:/$ cd

ubuntu@ubuntu:~$ pwd
/home/ubuntu

ubuntu@ubuntu:~$ cd /usr/bin

ubuntu@ubuntu:/usr/bin$ cd /home

ubuntu@ubuntu:/home$ cd -
/usr/bin

ubuntu@ubuntu:/usr/bin$ cd ~

ubuntu@ubuntu:~$ pwd
/home/ubuntu

ubuntu@ubuntu:~$ cd ./././.

ubuntu@ubuntu:~$ cd ../..

ubuntu@ubuntu:/$ pwd
/

ubuntu@ubuntu:/$
```

Observații:

- **cd ..** schimbă directorul în directorul părinte;

- . (punct) este o referință la directorul curent, deci **cd .** nu modifică directorul;
- ~ este echivalentul directorului home pentru utilizatorul curent (**cd** ~);
- **cd** apelat fără parametri schimbă directorul în directorul home;
- **cd -** schimbă directorul în directorul anterior.

4.3.2 Afișarea conținutului fișierelor

Conținutul unui fișier poate fi afișat pe ecran prin intermediul comenzii **cat**. Sintaxa acestei comenzi este **cat <nume fișier>**. Un exemplu de utilizare este:

```
1 ubuntu@ubuntu:~$ cat /etc/resolv.conf
2 # Generated by dhcpcd for interface eth0
3 search localdomain
4 nameserver 192.168.0.1
```

Această comandă afișează tot fișierul, inclusiv dacă acesta este mai mare de un ecran. Pentru a putea naviga prin output-ul unei comenzi (în cazul acesta, al comenzii **cat**), se pot folosi comenzile **more** sau **less**.

more permite navigarea doar intr-o singură direcție (de la început către sfârșitul fișierului) și câte un ecran odată, pe când **less** permite navigarea în ambele direcții, câte o linie, la fel ca un editor. **more** și **less** poartă numele de **paginatoare** (*pagere*). Interfața **less** este foarte asemănătoare cu cea a editorului Vim (vezi secțiunea 14.3).

```
1 ubuntu@ubuntu:~$ cat /etc/X11/xorg.conf | more
2
3 ubuntu@ubuntu:~$ cat /etc/X11/xorg.conf | less
```

4.3.3 Listarea conținutului unui director

Cea mai frecventă operație care se execută asupra directoarelor este listarea. Comanda utilizată este **ls**. În această secțiune, prin fișier înțelegem orice tip de fișier (inclusiv tipul "director"), mai puțin în cazurile când este menționat explicit.

Sintaxa comenzii pentru listare este **ls [opțiuni] [<cale>]**. În momentul în care lipsește calea, se realizează listarea conținutului directorului curent.

Opțiunile cele mai des folosite cu aceasta comandă sunt:

- `-l` afișează informații detaliate despre fiecare fișier/director (data modificare, dimensiune, utilizator, grup, drepturi de acces);
- `-a` afișează și fișierele care încep cu . (caracterul punct); în Unix, aceste fișiere sunt considerate de majoritatea interfețelor cu utilizatorul ca fiind fișiere ascunse;
- `-h` afișează dimensiunea fișierelor în format human-readable, respectiv dimensiunea în octeți este înlocuită cu dimensiunea în Kiloocteți/Megaocteți/Gigaocteți dacă depășește un anumit ordin de mărime.

Formatul utilizat la afișarea detaliată conține mai multe informații despre acel fișier:

```
ubuntu@ubuntu:~$ ls -l /home/myuser/myapp
-rwxr----- 1 myuser users 1176348 2007-03-03 20:28 /home/myuser/myapp
```

Semnificația coloanelor afișate este:

- primul caracter reprezintă tipul fișierului:
  - `-` = fișier normal
  - `d` = director
  - `p` = pipe
  - `b` = dispozitiv bloc
  - `c` = dispozitiv caracter
  - `l` = legătură simbolică
- urmează 3 grupuri de câte 3 caractere (`rwx`) care reprezintă drepturile de acces pentru utilizatorul `myuser` (care este deținătorul fișierului), drepturile de acces pentru utilizatorii care fac parte din grupul `users` (care deține fișierul), drepturile de acces pentru ceilați utilizatori; mai multe detalii despre drepturile de acces se găsesc în secțiunea 4.5;
- numărul de link-uri către fișier – `1`;
- utilizatorul ce deține fișierul - `myuser`;
- grupul de care aparține fișierul – `users`;
- dimensiunea fișierului – `1176348`;
- data fișierului – `2007-03-03`;
- ora fișierului – `20:28`;
- numele fișierului – `/home/myuser/myapp`.

Exemplele următoare prezintă câteva utilizări mai întâlnite pentru comanda **`ls`** care combină parametrii prezentați anterior:

```
ubuntu@ubuntu:~$ ls
Desktop

ubuntu@ubuntu:~$ ls ~
Desktop

ubuntu@ubuntu:~$ ls -l ~
total 0
drwxr-xr-x 2 ubuntu ubuntu 100 2007-09-03 13:15 Desktop

ubuntu@ubuntu:~$ ls -al
total 56
drwxr-xr-x 7 ubuntu ubuntu  580 2007-09-03 19:58 .
drwxr-xr-x 3 root   root     60 2007-09-03 06:14 ..
-rw------- 1 ubuntu ubuntu   34 2007-09-03 19:58 .bash_history
-rw-r--r-- 1 ubuntu ubuntu 2346 2007-09-03 06:14 .bashrc
drwxr-xr-x 2 ubuntu ubuntu  100 2007-09-03 13:15 Desktop
-rw-r--r-- 1 ubuntu ubuntu  566 2007-09-03 06:14 .profile
-rw------- 1 ubuntu ubuntu 9524 2007-09-03 17:40 .xsession-errors
```

```
20
21 ubuntu@ubuntu:~$ ls -alh
22 total 56K
23 drwxr-xr-x 7 ubuntu ubuntu  580 2007-09-03 19:58 .
24 drwxr-xr-x 3 root   root     60 2007-09-03 06:14 ..
25 -rw------- 1 ubuntu ubuntu   34 2007-09-03 19:58 .bash_history
26 -rw-r--r-- 1 ubuntu ubuntu 2.3K 2007-09-03 06:14 .bashrc
27 drwxr-xr-x 2 ubuntu ubuntu  100 2007-09-03 13:15 Desktop
28 -rw-r--r-- 1 ubuntu ubuntu  566 2007-09-03 06:14 .profile
29 -rw------- 1 ubuntu ubuntu 9.5K 2007-09-03 17:40 .xsession-errors
30
31 ubuntu@ubuntu:~$ ls ../../
32 bin    dev   initrd      media  proc  sbin  tmp  vmlinuz
33 boot   etc   initrd.img  mnt    rofs  srv   usr
34 cdrom  home  lib         opt    root  sys   var
35
36 ubuntu@ubuntu:~$ ls -a /
37 .    boot   etc     initrd.img  mnt   rofs  srv  usr
38 ..   cdrom  home    lib         opt   root  sys  var
39 bin  dev    initrd  media       proc  sbin  tmp  vmlinuz
```

Observații:

- dacă se dorește afișarea conținutului directorului curent, nu este necesară scrierea căii ca parametru pentru comanda **`ls`**;
- **`ls -al`** afișează lista detaliată a fișierelor din directorul home al utilizatorului curent, inclusiv fișierele ascunse (a se observă că și `.` și `..` sunt afișate;
- **`ls -alh`** afișează lista detaliată a fișierelor din directorul home al utilizatorului curent, utilizând formatul *human-readable* pentru afișarea dimensiunii fișierelor (a se observa dimensiunea fișierului `.xsession-errors`);
- **`ls ../..`** prezintă utilizarea lui **`ls`** cu parametru dat sub formă de cale relativă (directorul afișat este `/`);
- **`ls -a /`** prezintă fișierele ascunse ale directorului `/` – se observă că și acest director conține directoarele standard `.` și `...`

Folosind opțiunea `-R` se poate afișa arborele de directoare și fișiere care are ca rădăcină directorul specificat ca argument:

```
1 ubuntu@ubuntu:~$ ls -R dir1
2 dir1:
3    dir2  fisier1  fisier2
4 dir1/dir2:
5    fisier3  fisier4
```

Pentru mai multe opțiuni se poate consulta **`ls --help`** sau **`man ls`**.

4.3.4 Crearea fișierelor/directoarelor

Pentru a crea orice entitate pe un sistem de fișiere, se utilizează o serie de comenzi, prezentate în tabelul 4.5.

Primul mod de creare a unui fișier este utilizând comand comanda **`touch`**. A doua metodă se bazează pe o funcționalitate bash numită redirectare în fișiere (vezi

Tabelul 4.5: Comenzi pentru crearea fișierelor

Entitate	Comandă
Fișier normal	`touch <nume_fișier>`, `> <nume_fișier>`
Director	`mkdir <nume_director>`
Legături (link-uri)	`ln -s <destinatie> [<nume_legătură>]`
Pipe-uri cu nume	`mkfifo <nume_pipe>`

secțiunea 4.4). Pe scurt, ceea ce realizează comanda a 2-a este redirectarea ieșirii unei comenzi (nule) către un fișier, creându-se astfel un fișier gol.

```
ubuntu@ubuntu:~$ touch fisier1

ubuntu@ubuntu:~$ > fisier2

ubuntu@ubuntu:~$ mkdir dir1

ubuntu@ubuntu:~$ mkfifo fifo1

ubuntu@ubuntu:~$ ls -l
total 0
drwxr-xr-x 2 mircea users 40 2007-09-20 10:26 dir1
prw-r--r-- 1 mircea users  0 2007-09-20 10:30 fifo1
-rw-r--r-- 1 mircea users  0 2007-09-20 10:26 fisier1
-rw-r--r-- 1 mircea users  0 2007-09-20 10:26 fisier2
```

O altă întrebuințare a comenzii **`touch`** este aceea a actualizării datei ultimei modificări și a datei ultimei accesări. Folosind opțiunea `-m` se va actualiza doar data ultimei modificări iar folosind opțiunea `-a` se va actualiza doar data ultimei accesări. Folosind comanda **`touch`** fără aceste opțiuni va duce la modificarea ambelor date. Optiunea `-r` poate fi folosită pentru a prelua informațiile legate de timp de la alt fișier.

În exemplul de mai jos, se observă modificarea timpului de acces al fișierului de la valoarea `17:03` la `17:11`.

```
ubuntu@ubuntu:~$ ls -al
-rw-r--r--  1 root     root       15 2009-08-21 17:03 fisier1

ubuntu@ubuntu:~$ touch fisier1

ubuntu@ubuntu:~$ ls -al
-rw-r--r--  1 root     root       15 2009-08-21 17:11 fisier1
```

Exemple privind crearea de link-uri sunt prezentate în secțiunea 4.2.1.

4.3.5 Copiere/mutare/redenumire/ștergere

Copierea și mutarea sunt operații care necesită 2 parametri:

- sursă de unde se copiază/mută;
- destinația unde se copiază/mută.

Copierea

Copierea unui fișier sau director se realizează cu ajutorul comenzii **cp**. Sintaxa comenzii este **cp [opțiuni] <sursa> <destinatie>**. Opțiunile cele mai folosite pentru copiere sunt:

- `-R` – copiere recursivă (copiază și copiii directoarelor, presupunând că există așa ceva în sursă);
- `-p` – copiere cu păstrare a tuturor atributelor (permisiuni, dată);
- `-u` – copiază doar dacă fișierul sursă este mai nou decât fișierul destinație sau dacă fișierul destinație lipsește.

```
1 ubuntu@ubuntu:~$ touch f1.txt
2
3 ubuntu@ubuntu:~$ ls
4 f1.txt
5
6 ubuntu@ubuntu:~$ cp f1.txt f2.txt
7
8 ubuntu@ubuntu:~$ ls
9 f1.txt  f2.txt
```

Mutarea

Mutarea unui fișier sau director se realizează cu ajutorul comenzii **mv**. Sintaxa comenzii este **mv [opțiuni] <sursa> <destinatie>**. Implicit, mutarea este recursivă și păstrează atributele fișierelor. În cazul în care sursa și destinația se găsesc pe aceeași partiție, la o mutare, se schimbă doar părintele fișierului sau directorului care se mută. O operație de mutare este, astfel, mai puțin costisitoare decât o operație de copiere.

```
1  ubuntu@ubuntu:~$ pwd
2  /tmp/q
3
4  ubuntu@ubuntu:~$ ls
5  f1.txt  f2.txt
6
7  ubuntu@ubuntu:~$ mv f1.txt /tmp/r
8
9  ubuntu@ubuntu:~$ ls
10 f2.txt
11
12 ubuntu@ubuntu:~$ ls /tmp/r/
13 f1.txt
```

Redenumirea

Redenumirea este, de fapt, o mutare și se realizează cu ajutorul comenzii **mv**, în următoarele conditii:

- în cazul în care sursa **este un fișier** și destinația **este un fișier**, se realizează copierea/mutarea fișierului cu schimbarea numelui sursei;

- în cazul în care sursa **este un director** și destinația **nu există ca director**, se realizeaza copierea/mutarea directorului cu schimbarea numelui.

```
ubuntu@ubuntu:~$ ls
f2.txt

ubuntu@ubuntu:~$ mv f2.txt f3.txt

ubuntu@ubuntu:~$ ls
f3.txt
```

Stergerea fișierelor/directoarelor

În Linux, comanda cea mai utilizata pentru ștergerea fișierelor și directoarelor este **`rm`**. Sintaxa comenzii este **`rm [opțiuni] <cale>`**.

Una dintre cele mai folosite opțiuni este `-r/-R`, utilizată pentru a șterge recursiv un director. Aceasta opțiune trebuie folosită cu atenție pentru ca poate avea rezultate negative dacă directorul care este șters conține informații utile. O eroare celebră este utilizarea comenzii **`rm -rf /`**, care șterge recursiv totul începând cu directorul rădăcină, forțând ștergerile (datorită opțiunii `-f`).

Pentru ștergerea directoarelor goale se poate folosi și `rmdir`.

4.3.6 Arhivarea fișierelor și dezarhivarea

> **Arhivarea** este procedeul prin care mai multe fișiere și directoare sunt adunate la un loc (într-un singur fișier), realizând eventual și o reducere a dimensiunii prin eliminarea datelor care se repetă și înlocuirea lor cu o serie de codificări.

În general, în cadrul procesului de arhivare se creează un dicționar cu secvențe de octeți frecvente și o codificare a lor mai scurtă (pe mai puțini octeți). Arhiva conține atât dicționarul cât și conținutul fișierelor dar, în loc de secvențele frecvente lungi, se regăsesc referințe către dicționar. Dacă fișierele conțin multe date care se repetă, dimensiunea unei arhive scade considerabil.

După cum se poate identifica și din descrierea anterioară, există două etape mai importante în ceea ce privește manevrarea fișierelor:

- concatenarea (lipirea) fișierelor într-un fișier mai mare, din care să se poată extrage toate fișierele și informațiile despre ele;
- compresia fișierului mare, astfel încât să se reducă dimensiunea lui dar fără a se pierde din informații.

În Linux există utilitare de arhivare care se ocupă ori de una dintre etape ori de ambele etape.

Comanda **`tar`** se ocupă de prima etapă. Numele utilitarului este o abreviere a *tape archive*, întrucât, la începutul utilizării sale, rezultatul comenzii era transferat pe benzi magnetice.

Un fișier `.tar` conține fișiere necomprimate împreună cu informații despre modul de extragere al acestora (spre exemplu: de unde până unde se găsește un fișier în cadrul arhivei). Din acest motiv, un fișier `.tar` este de obicei mai mare decât suma tuturor dimensiunilor fișierelor ce sunt incluse în el.

Pentru a crea o arhiva `.tar`, se utilizează comanda:

```
1 ubuntu@ubuntu:~$ tar cvf nume_arhiva.tar <cale>
```

cu parametrii:

- `c` creează arhiva;
- `v` afișează ce anume se arhivează;
- `f nume_arhiva.tar` specifică numele arhivei;
- `<cale>` precizează directorul/directoarele/fișiere care vor fi arhivate.

Pentru a dezarhiva o arhivă `.tar`, se utilizează:

```
1 ubuntu@ubuntu:~$ tar xvf nume_arhiva.tar [-C <cale_destinatie>]
```

unde:

- `x` dezarhivează (*eXtract*);
- `v` afișează ce anume se dezarhivează;
- `f nume_arhiva.tar` precizează numele arhivei care se dezarhivează;
- `-C <cale_destinatie>` specifică, opțional, locul unde se realizează dezarhivarea.

`f` și `nume_arhiva.tar` se consideră un singur parametru; din acest motiv, de fiecare dată când se folosește opțiunea `f` pentru a indica un fișier arhivă, aceasta trebuie să apară ultima în lista de opțiuni, fiind urmat imediat de numele fișierului arhivă.

Pentru a realiza compresia unui fișier, două utilitare sunt folosite preponderent în lumea Unix:

- **`gzip`**, mai rapid dar cu o rată de compresie mai mică;
- **`bzip2`**, mai lent dar cu o rată de compresie mai mare.

Comanda **`tar`** poate utiliza direct unul dintre programele de comprimare menționate anterior folosind parametrul `z` pentru **`gzip`** sau parametrul `j` pentru **`bzip2`**, ca în tabelul 4.7

Tabelul 4.6: Comprimarea și decomprimarea folosind gzip și bzip2

	Comprimare	**Decomprimare**
gzip	**`tar czvf fisier.tar.gz dir/`**	**`tar xzvf fisier.tar.gz`**
bzip2	**`tar cjvf fisier.tar.bz2 dir/`**	**`tar xjvf fisier.tar.bz2`**

Pe lângă opțiunile de compresie/arhivare se mai pot folosi și alte opțiuni. Printre cele mai utile se numără opțiunea `--preserve`, care impune păstrarea drepturilor de acces în momentul arhivării/dezarhivării.

Câteva exemple de utilizare a comenzii **tar** pentru comprimare și decomprimare, în conjuncție cu **bzip2** și **gzip** sunt prezentate mai jos:

```
ubuntu@ubuntu:/tmp/q$ ls
f1.txt  f2.txt  f3.txt

ubuntu@ubuntu:/tmp/q$ tar cvf arhiva.tar f*.txt
f1.txt
f2.txt
f3.txt

ubuntu@ubuntu:/tmp/q$ file arhiva.tar
arhiva.tar: POSIX tar archive (GNU)

ubuntu@ubuntu:/tmp/q$ gzip arhiva.tar

ubuntu@ubuntu:/tmp/q$ ls
arhiva.tar.gz  f1.txt  f2.txt  f3.txt

ubuntu@ubuntu:/tmp/q$ file arhiva.tar.gz
arhiva.tar.gz: gzip compressed data, was "arhiva.tar", from Unix, last
modified: Thu Sep 20 11:37:30 2007

ubuntu@ubuntu:/tmp/q$ ls
arhiva.tar.gz  f1.txt  f2.txt  f3.txt

ubuntu@ubuntu:/tmp/q$ mv arhiva.tar.gz fisier.txt

ubuntu@ubuntu:/tmp/q$ file fisier.txt
fisier.txt: gzip compressed data, was "arhiva.tar", from Unix, last
modified: Thu Sep 20 11:37:30 2007

ubuntu@ubuntu:/tmp/q$ ls
f1.txt  f2.txt  f3.txt

ubuntu@ubuntu:/tmp/q$ tar czvf new.tar.gz f*
f1.txt
f2.txt
f3.txt

ubuntu@ubuntu:/tmp/q$ ls
f1.txt  f2.txt  f3.txt  new.tar.gz

ubuntu@ubuntu:/tmp/q$ file new.tar.gz
new.tar.gz: gzip compressed data, from Unix, last modified: Thu Sep 20
11:40:02 2007

ubuntu@ubuntu:/tmp/q$ mkdir tmp

ubuntu@ubuntu:/tmp/q$ mv new.tar.gz tmp/

ubuntu@ubuntu:/tmp/q$ cd tmp/

ubuntu@ubuntu:/tmp/q/tmp$ tar xvf new.tar.gz
f1.txt
f2.txt
f3.txt
ubuntu@ubuntu:/tmp/q/tmp$ ls
f1.txt  f2.txt  f3.txt  new.tar.gz
```

4.3.7 Backup

Backup-ul este utilizat pentru a păstra într-un loc separat o copie a datelor. Această copie se utilizează pentru a reface datele în cazul în care suportul original nu mai poate fi folosit din orice motiv. Este una dintre cele mai utile acțiuni asupra datelor pe care toată lumea știe că ar fi bine să o facă dar nu o face. Câteva metode de backup au fost deja prezentate. Tabelul următor prezintă câteva metode de backup și situațiile când sunt ele potrivite.

Tabelul 4.7: Comprimarea și decomprimarea folosind gzip și bzip2

Metodă	Descriere
`tar+gzip/bzip2`	Metodă foarte simplu de aplicat. Devine greu de folosit pentru dimensiuni mari de date. Permite comprimarea datelor.
`dd`	Metoda simplu de folosit și independentă de sistemul de fișiere. Permite păstrarea intactă a structurii sistemului de fișiere. Inflexibilă când vine vorba de recuperarea datelor. Utilă pentru cantități mari de date.
`rsync`[1]	E asemănătoare comenzii `cp` dar la care s-a adăugat suport de sincronizare intre mai multe sisteme de calcul. Permite replicarea structurii de fișiere (inclusiv permisiuni) între 2 sisteme de calcul.
`rdiff-backup`[2]	Este un wrapper peste `rsync`. Adaugă suport pentru backup-uri incrementale. La un moment dat se realizează un backup complet pentru un director (asemănător rsync-ului). Backup-urile incrementale salvează doar modificările ce s-au făcut de la ultimul backup până în prezent, indiferent de tipul backup-ului. În acest fel se poate reveni la orice stare anterioara, în măsura în care s-a realizat cel puțin un backup incremental la acea stare.

4.4 Redirectări de comenzi

4.4.1 Descriptorii de fișier

Fiecare program în execuție utilizează 3 fișiere speciale pentru a interacționa cu utilizatorul:

- **stdin** (*standard input*) – fișier care reprezintă locul de unde programul își citește datele de intrare (de obicei tastatura);
- **stdout** (*standard output*) – fișier care reprezintă locul unde programul scrie datele de ieșire (de obicei acesta este consola curentă);
- **stderr** (*standard error*) – fișier care reprezintă locul unde programul scrie mesajele de eroare (de obicei acesta este consola curentă).

În C/C++, există 3 variabile de tipul `FILE *` cu numele `stdin`, `stdout` și `stderr` care sunt inițializate și disponibile în orice moment. Ele îndeplinesc rolurile descrise anterior și pot fi folosite ca pe orice alta variabila de tipul `FILE *`.

Fiecare fișier deschis de o aplicație Linux are asociat un indice. Acest indice se numește descriptor de fișier. Fișierele descrise anterior au descriptorii de fișier următori:

- stdin – 0
- stdout – 1
- stderr – 2

Orice fișier deschis de aplicație va avea un descriptor mai mare sau egal cu 3.

4.4.2 Redirectări

Există situații când, pentru o aplicație, utilizatorul dorește să modifice intrarea sau ieșirea – spre exemplu, utilizatorul dorește ca ieșirea unui program să se facă într-un fișier și nu în consola curentă, sau intrarea unui program să fie un fișier și nu tastatura. Acest lucru se poate face doar lucrând la nivelul descriptorilor.

Cazurile întâlnite sunt prezentate în Tabela 4.8.

Tabelul 4.8: Comenzi de redirectare

sursă	**destinație**	**exemplu comandă**
intrare (stdin)	fișier	`./program < f_in`
ieșire (stdout)	fișier	`./program > f_out`
eroare (stderr)	fișier	`./program 2> f_err`
eroare (stderr)	ieșire (stdout)	`./program 2>&1`
eroare și ieșire	fișier	`./program 2>&1 > f_out_err` sau `./program &> f_out_err`

Se observă că pentru a redirecta ieșirea/intrarea se folosesc semnele `>` (*mai mare*) pentru ieșire (către exteriorul programului) și `<` (*mai mic*) pentru intrare (către program). Pentru redirectarea ieșirii de erori se indică descriptorul de fișier `2`. Pentru a redirecta către ieșirea standard (*stdout*), se folosește sintaxa `&1`.

Pentru redirectarea ieșirii de erori și iesirii standard către un fișier, se redirectează întâi ieșirea de erori către ieșirea standard folosind `2>&1` dupa care se redirectează ieșirea standard către un fișier `>fișier_iesire_și_erori`. **Bash** oferă o sintaxă simplificată în cadrul operatorului `&>`.

Ca exemplu de redirectare a intrării, în comanda anterioară se trimite conținutul fișierului `continut_mail` ca intrare pentru comanda **`mail`**. Se va trimite un mesaj utilizatorului **`user1`** cu informațiile din fișier.

```
1 ubuntu@ubuntu:~$ mail user1 < continut_mail
```

În exemplul de mai jos se scrie lista de fișiere și directoare din directorul curent în fișierul `listare`.

```
ubuntu@ubuntu:~$ ls > listare

ubuntu@ubuntu:~$ cat listare
fisier1
```

Pentru a adăuga rezultatul comenzii date la sfârșitul fișierului se folosește operatorul `>>`:

```
ubuntu@ubuntu:~$ date >> listare

ubuntu@ubuntu:~$ cat listare
fisier1
Sat Aug 22 16:07:50 EEST 2009
```

În listingul de mai jos se încearcă copierea unui fișier fără specificarea destinației. Această comandă generează o eroare care este scrisă în fișierul `erori`:

```
ubuntu@ubuntu:~$ cp fisier1 2> erori

ubuntu@ubuntu:~$ cat erori
cp: missing destination file operand after 'fisier1'
Try 'cp --help' for more information.
```

Pentru a adăuga noi mesaje de eroare la sfârșitul fișierului se folosește operatorul `2>>`:

```
ubuntu@ubuntu:~$ cp fisier4 2>> erori

ubuntu@ubuntu:~$ cat erori
cp: missing destination file operand after 'fisier1'
Try 'cp --help' for more information.

cp: missing destination file operand after 'fisier4'
Try 'cp --help' for more information.
```

Comanda de mai jos scrie și rezultatele și erorile generate de comanda ls în fișierul specificat:

```
ubuntu@ubuntu:~$ ls fisier1 fisier4 2>&1 > rezultate
```

sau mai simplu:

```
ubuntu@ubuntu:~$ ls fisier1 fisier4 >& rezultate

ubuntu@ubuntu:~$ cat rezultate
ls: cannot access fisier4: No such file or directory
fisier1
```

Tabelul 4.9 prezintă câteva exemple de redirectări folosind fișiere speciale:

4.5 Drepturi de acces

O primă măsură de protecție a datelor o reprezintă drepturile de acces la fișiere. Atât timp cât un utilizator nu are drepturi de administrator pe un anumit computer, acel utilizator se supune drepturilor de acces la fișiere.

Tabelul 4.10 prezintă drepturi ce pot fi configurate pentru un fișier, fără a particulariza la un anumit tip de sistem de fișiere.

Tabelul 4.9: Comenzi de redirectare care folosesc fișiere speciale

Comandă	Efect
`./program 2>/dev/null`	mesajele de la ieșirea de erori nu sunt afișate
`./program 2>&1 >/dev/null`	niciun mesaj nu este afișat
`> fout`	creează un fișier gol cu numele `f_out`
`cat /dev/null > f_out`	creează un fișier cu același conținut ca `/dev/null`, adică un fișier gol

Tabelul 4.10: Drepturi de acces

Drept	Descriere
citire	dreptul de a deschide și citi conținutul unui fișier
scriere	dreptul de a scrie într-un fișier
execuție	dreptul de a executa un fișier (aplicație) sau, pentru directoare, dreptul de a intra într-un director
modificare	dreptul de a modifica datele dintr-un fișier existent
ștergere	dreptul de a sterge un fișier

4.5.1 Utilizatori și grupuri de utilizatori vs. liste de acces

Există două metode mai întâlnite pentru definirea drepturilor de acces la fișiere:

- drepturi de acces la nivel de utilizator/grup.
- liste de acces – *Access Control Lists* (ACL).

Prima metoda (drepturi de acces la nivel de utilizator/grup) constă în definirea unor drepturi pentru următoarele entități:

- posesorul unui fișier (*user*);
- grupul care deține fișierul (*group*);
- toți ceilalți utilizatori (*others*).

Un utilizator se poate afla în mai multe grupuri (vezi secțiunea 3.1). Această metodă este cea mai folosită cale pentru definirea drepturilor de acces. Metodă oferă un nivel de protecție suficient pentru majoritatea situațiilor, ocupând un spațiu limitat.

În cadrul sistemului bazat pe liste de acces, unui fișier i se pot asocia mai mulți utilizatori și/sau grupuri de utilizatori. Pentru fiecare dintre aceste entități pot fi configurate mai multe tipuri de drepturi. Această variantă poate fi privită ca o extindere a sistemului cu drepturi de acces prezentat anterior. Complexitatea poate face ca sistemul să fie destul de greu de întreținut.

Fiecare sistem de fișiere oferă un set de drepturi ce pot fi modificate pentru fiecare fișier în parte. Ca diferențe notabile între sistemele de fișiere se pot aminti următoarele:

- FAT32 nu oferă suport pentru drepturi de acces la fișiere; există doar posibilitatea de a marca un fișier ca read-only; orice utilizator poate să schimbe acest drept;

- majoritatea sistemelor de fișiere în mediile Unix (inclusiv Mac OS X) au suport pentru drepturi de acces bazat pe utilizator/grup și, folosind o extensie, pot fi extinse cu liste de acces;
- NTFS are un sistem foarte avansat de drepturi de acces, bazat pe liste de acces (vezi secțiunea 10.6.1); pentru fiecare entitate adăugată în lista unui anumit fișier pot fi configurate mai multe drepturi.

4.5.2 Clasificarea drepturilor de acces

Sistemele de fișiere din mediile Unix care au ca implementare standard drepturile de acces la nivel user/grup au la baza următorul set de drepturi de acces:

- **citire** (*read*) – deschidere și citire de fișiere;
- **scriere** (*write*) – creare și scriere de fișiere;
- **execuție** (*execute*) – execuție fișiere/intrare în directoare.

În aceste sisteme de fișiere exista 3 entități pentru care poate fi stabilită orice combinație a drepturilor definite mai sus:

- **utilizator** (*user*) – posesorul fișierului;
- **grup** (*group*) – grupul de care aparține fișierul;
- **alții** (*others*) – utilizatorii care nu fac parte din grupul fișierului și nici nu sunt posesori.

4.5.3 Vizualizarea drepturilor de acces

Pentru a vedea drepturile de acces pentru un anumit fișier este suficientă utilizarea comenzii **`ls`** cu parametrul `-l`. Spre exemplu, pentru a vedea drepturile de acces asupra unui fișier, transmitem ca parametru comenzii **`ls`** direct numele fișierului:

```
1 ubuntu@ubuntu:~$ ls -l fisier.txt
2 -rwxr----- 1 ubuntu users 0 2007-09-20 12:47 fisier.txt
```

Drepturile de acces sunt date de primele 10 caractere din ieșirea comenzii **`ls`**:

- primul caracter reprezintă tipul fișierului:
  - `-` = fișier obișnuit
  - `d` = director
  - `p` = pipe
  - `b` = dispozitiv bloc
  - `c` = dispozitiv caracter
  - `l` = legătură

- următoarele 3 caractere (`rwx`) reprezintă drepturile de acces pentru utilizatorul `ubuntu`, care este deținătorul fișierului; se observă că utilizatorul are toate drepturile (scriere/citire/execuție);
- următoarele 3 caractere (`r--`) reprezintă drepturile de acces pentru utilizatorii care fac parte din grupul `users`; se observă că este prezent dreptul de citire dar lipsesc drepturile de scriere și execuție;
- următoarele 3 caractere (`---`) reprezintă drepturile de acces pentru utilizatorii care nu sunt `ubuntu` și nici nu fac parte din grupul `users`; acest utilizatori nu au niciun drept.

Există două moduri de reprezentare a drepturilor:

- în formă **numerică**: pentru fiecare entitate exista o cifră în baza 8 care descrie drepturile, câte un bit pentru fiecare drept;
- în formă **literală**: drepturile sunt referite direct prin inițiala lor, pentru fiecare tip de entitate.

Astfel, pentru exemplul de mai sus, avem drepturile:

- `rwxr-----` în forma listing;
- `111100000` în forma binară;
- `740` în forma octală;
- `u=rwx,g=r` în forma literală.

Mai multe informații despre securitatea fișierelor și modul de schimbare a drepturilor de acces se găsesc în secțiunea 10.2.4.

4.6 Căutarea fișierelor

4.6.1 Comanda find

Comanda **`find`** folosește abordarea brute-force pentru găsirea fișierelor căutand în arborele de directoare. Comanda pune la dispoziție un set extins de criterii de căutare, cum ar fi numele fișierului, utilizator, grup, tip, permisiuni, dimensiune, dată, și altele. Exemple de criterii de căutare sunt prezentate în continuare specificând opțiunea corespunzătoare a comenzii **`find`**:

- `-name` – căutare după numele fișierelor
- `-perm` – căutare după permisiunile fișierelor
- `-size` – căutare după dimensiunea fișierelor

Comanda următoare caută toate fișierele care au numele `stat`, în mod recursiv în directorul `/usr`:

```
1 ubuntu@ubuntu:~$ find /usr -name stat
2 /usr/src/linux-headers-2.6.24-19-generic/include/config/cpu/freq/stat
3 /usr/bin/stat
```

Următoarea comandă caută toate fișierele care au permisiunile `644`:

```
ubuntu@ubuntu:~$ find /usr -perm 644
/usr/local/include/glib-2.0/glib.h
/usr/local/include/glib-2.0/gobject/gsourceclosure.h
/usr/local/include/glib-2.0/gobject/gtypeplugin.h
[...]
```

Pentru a căuta fișierele cu dimensiunea mai mare de 500 KB se poate folosi una dintre următoarele comenzi:

```
ubuntu@ubuntu:~$ find /usr -size +500000

ubuntu@ubuntu:~$ find /usr -size +500k
/usr/local/lib/libgio-2.0.so.0.1800.4
/usr/local/lib/libgobject-2.0.so.0.1800.4
/usr/local/lib/libglib-2.0.so.0.1800.4
[...]
```

4.6.2 Comanda locate

Comanda **`locate`** foloseste o baza de date locală în care sunt indexate toate fișierele. Comanda **`locate`** este mai rapida decât comanda **`find`**, dar pune la dispoziție un singur criteriu de căutare: numele fișierului. Un alt dezavantaj este faptul că baza de date trebuie reactualizată periodic pentru a conține informații despre fișierele noi create în sistem. Actualizarea se realizează cu ajutorul comenzii **`updatedb`**.

Comanda întoarce o listă cu toate fișierele ale cărui nume conține șirul de caractere precizat ca argument. În exemplul următor se caută toate fișierele care conțin `pwd`:

```
ubuntu@ubuntu:~$ locate pwd
/bin/pwd
/etc/.pwd.lock
/sbin/unix_chkpwd
/usr/bin/pwdx
/usr/include/pwd.h
[...]
```

Unele distribuții Linux folosesc comanda **`slocate`** în locul comenzii locate. Avantajul comenzii **`slocate`** este acela că nu permite afișarea fișierelor din directoarele în care utilizatorul nu are drepturi de acces.

4.6.3 Comanda whereis

Această comandă poate fi folosită pentru a căuta într-un set restrâns de locații din sistem, de exemplu directoarele cu fișiere binare, directoarele cu biblioteci sau directoarele cu pagini de manual. Comanda **`whereis`** nu poate fi folosită pentru a căuta în directoarele utilizatorului. Comanda va căuta toate fișierele care încep cu șirul de caractere precizat ca argument. De exemplu, pentru a localiza comanda **`ls`** folosim următoarea comandă:

```
ubuntu@ubuntu:~$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.gz
```

Comanda va afișa calea către executabil dar și calea către pagina de manual a comenzii **ls**.

4.6.4 Comanda which

Folosind comanda `which` se poate obține calea către executabilele care pot fi rulate din linia de comandă. De exemplu, pentru a afla calea către comanda **chmod** se va folosi comanda următoare:

```
1 ubuntu@ubuntu:~$ which chmod
2 /bin/chmod
```

În mod implicit, comanda **which** returnează doar prima potrivire găsita, iar pentru a afișa lista completa a potrivirilor se va folosi opțiunea `-a`.

4.6.5 Comanda type

Această comandă poate fi folosită pentru a determina modul de interpretare a unei comenzi, de exemplu comandă integrată în shell, comandă externă sau alias.

Un exemplu de comandă integrată in shell este:

```
1 ubuntu@ubuntu:~$ type cd
2 cd is a shell builtin
```

Pentru o comandă externă rezultatul comenzii type este calea către executabil:

```
1 ubuntu@ubuntu:~$ type cat
2 cat is /bin/cat
```

În cazul unui alias este afișată comanda echivalentă:

```
1 ubuntu@ubuntu:~$ type ls
2 ls is aliased to 'ls --color'
```

4.7 Tipuri de sisteme de fișiere

În timp au apărut mai multe tipuri de sisteme de fișiere. În mediul open-source s-au dezvoltat și încă se dezvoltă cele mai multe dintre ele. Câteva dintre cele mai importante sisteme de fișiere utilizate în prezent sunt menționate în tabelul 4.11, împreună cu sistemele de operare în care se întâlnesc.

Prin *nativ* se înțelege că suportul este oferit prin drivere ce însoțesc sistemul de operare (în cazul Linux însoțesc kernel-ul).

Sistemele de fișiere se pot clasifica și după locul unde se găsesc datele. Tabelul 4.12 prezintă succint această clasificare.

Tabelul 4.11: Sisteme de operare și sisteme de fișiere suportate

Sistem de fișiere	Windows	Linux	Mac OS X
FAT32	nativ	nativ	nativ
NTFS	nativ (după WinNT)	prin ntfs-3g	prin ntfs-3g
ext2/ext3	driver third-party	nativ	–
ReiserFS	aplicație third-party	nativ	–
HFS+	–	nativ	nativ
ISO9660	nativ	nativ	nativ
UDF	nativ	nativ	nativ

Tabelul 4.12: Clasificarea sistemeleor de fișiere după suportul datelor

Tip	Exemplu	Descriere
sisteme de fișiere cu suport fizic	FAT32, NTFS, Ext3, ReiserFS, HFS+	se regăsesc de obicei pe un mediu de stocare
sisteme de fișiere virtuale	procfs, devfs, SSHFS	conțin fișiere/date generate de SO (informații despre sistem) sau de o altă componentă software (alte surse)
sisteme de fișiere pentru rețea	NFS, SMB	utilizate pentru accesul la fișiere aflate în rețea

4.7.1 Integritatea datelor

Un sistem de fișiere trebuie să asigure integritatea datelor inclusiv și în cazul unei căderi (failure). De cele mai multe ori, această funcționalitate este implementată prin jurnalizare.

> **Jurnalizarea** este activitatea prin care se realizează un jurnal cu toate modificările efectuate asupra unui sistem de fișiere. Fiecare operație care se execută asupra sistemului de fișiere este întâi scrisă în jurnal. Operația poate presupune modificări în mai multe locuri din sistemul de fișiere.

Driver-ul sistemului de fișiere execută la un anumit interval de timp operațiile din jurnal pentru a actualiza starea discului. O operație se șterge din jurnal doar după ce toate modificările asociate ei au fost executate. În acest fel, în cazul apariției unei căderi (întrerupere de curent etc.), se pot executa operațiile din jurnal care au rămas neexecutate (operațiile începute și care au fost întrerupte sunt încă prezente în jurnal și vor fi reexecutate în întregime).

4.7.2 Alegerea unui sistem de fișiere

În momentul în care trebuie ales un sistem de fișiere, cele mai utilizate criterii de selecție sunt:

- **disponibilitatea** – dacă se poate folosi sistemul respectiv de fișiere în sistemul de operare ales (sau în mai multe sisteme de operare);

- **gradul de siguranță** – majoritatea sistemelor de fișiere existente în prezent au jurnalizare;
- **restricții speciale** date de modul de organizare a datelor în sistemul de fișiere; de exemplu, la FAT32 se folosesc doar 32 biți pentru a stoca dimensiunea unui fișier, deci dimensiunea maximă a unui fișier este 4GB (mai puțin decât o imagine de DVD, fapt ce îl face nepractic pentru o parte din aplicațiile multimedia);
- **optimizări de performanță**, realizate spre exemplu în funcție de tipul suportului de stocare, în funcție de dimensiunea fișierelor etc.

Tabelul 4.13 realizează o analiză sumară a caracteristicilor sistemelor de fișiere.

Tabelul 4.13: Caracteristici ale sistemelor de fișiere mai cunoscute

Tip sistem de fișiere	**Sisteme de operare**	**Dimensiune maximă fișier**	**Jurnalizare**	**Observații**
FAT32	Windows / Linux / Mac OS X	4 GB	nu	cel mai folosit sistem de fișiere – întâlnit în mod special pe USB stick-uri, fără drepturi de acces; Windows-ul limitează la creare dimensiunea unei partiții la 32GB, dar poate citi partiții mai mari realizate și formatate cu aplicații third-party folosit pentru a asigura compatibilitatea cu dispozitive sau sisteme mai vechi
NTFS	Windows / Linux / Mac OS X	16TiB	da	singurul sistem de fișiere pentru Windows recomandat de Microsoft; singurul sistem de fișiere pentru Windows cu securitate
ext2/ext3	Linux / Windows[1]	16GiB – 64TiB	ext2-nu; ext3-da	sistemul de fișiere considerat cel mai stabil datorită unei istorii de dezvoltare foarte lungi; ext3 e compatibil cu versiunea anterioară ext2, aduce jurnalizare față de ext2
ReiserFS	Linux	8 TiB	da	cunoscut pentru o viteză mai mare decât ext3-ul la lucrul cu multe fișiere de dimensiune mică
HFS+	Mac OS X / Linux	16 EiB	da[2]	–
ISO9660	Windows / Linux / Mac OS X	în funcție de implementare	nu	sistem de fișiere utilizat în principal pe CD-uri, organizare internă concepută pentru ca datele să fie citibile ușor
UDF	Windows / Linux / Mac OS X	16 EiB	da	sistem de fișiere utilizat în principal pe mediile optice, cu suport atât pentru scriere cât și pentru citire

4.7.3 Adresarea într-un sistem de fișiere

Din punct de vedere fizic, fișierele ocupă pe dispozitivul de stocare un spațiu bine definit. Utilizatorul nu lucrează direct cu dispozitivul de stocare și nu poate adresa un fișier după poziția fizică a acestuia pe dispozitivul de stocare. S-a introdus astfel adresarea fișierelor folosind nume. În funcție de tipul sistemului de fișiere, există o serie de restricții asupra acestor nume.

Majoritatea fișierelor au și ceea ce se numește "extensie". Aceasta extensie se regăsește după numele fișierelor și este utilizată pentru identificarea tipului acestora (exemplu format `nume.ext`. În majoritatea cazurilor această extensie are 3 caractere. Sistemele de operare de tip Windows utilizează extensia pentru a asocia un tip de fișier cu o anumită aplicație care poate deschide acel tip de fișier. În mediile Unix, extensia de obicei lipsește. Identificarea tipului fișierelor se realizează în general pe baza conținutului fișierelor.

În Unix, comanda utilizată pentru a descoperi tipul fișierelor este **`file`**:

```
1 ubuntu@ubuntu:~$ file /boot/kernel26.img
2 /boot/kernel26.img: gzip compressed data, from Unix, last modified: Sat
  Mar 3 20:28:37 2007, max compression
```

Numele fișierelor poate fi scris cu un anumit set de caractere. În prezent, majoritatea sistemelor de fișiere folosesc standardul Unicode (de obicei UTF-8 sau UTF-16) pentru codificarea caracterelor. În tabelul 4.14 sunt prezentate câteva sisteme de fișiere și restricțiile de nume asociate lor.

Dupa cum s-a menționat și la începutul acestui capitol, directoarele se utilizează pentru o mai bună organizare a fișierelor. Într-un sistem de fișiere modern, fișierele și directoarele se regăsesc într-o structură ierarhică – sub forma unui arbore. Acest arbore are un director părinte din care se poate ajunge în orice loc în sistemul de fișiere.

Calea către un fișier este formată dintr-o înșiruire de directoare, separate de caractere speciale. În mediile Unix separatorul este caracterul / (*slash*). În mediile Windows, separatorul este caracterul \ (*backslash*).

4.8 Lucrul cu sistemele de fișiere

Această secțiune tratează acțiunile cele mai cunoscute la nivelul unui sistem de fișiere. Acțiunile cele mai frecvente sunt:

- crearea unui sistem de fișiere;
- montarea unui sistem de fișiere;
- repararea unui sistem de fișiere.

Printre acțiunile rare dar care pot prezenta interes trebuie menționate:

- crearea unei imagini pentru un sistem de fișiere;
- ștergerea unui sistem de fișiere (din diferite motive).

Tabelul 4.14: Restricții asupra numelor fișierelor

Sistem de fișiere	Case sensitive	Lungime nume	Lungime cale	Observații
FAT32	nu	`8.3` sau 255 caractere cu LFN[1]	nedefinit	initial limitat la 8 caractere numele fișierului și 3 caractere extensia; FAT32 a capatat suport pentru nume lungi (LFN) pentru un fișier; FAT32 reține atât numele lung cât și versiunea scurta în format 8.3
NTFS	da[2]	254 caractere + .	32767 octeți	orice caracter Unicode poate fi folosit, mai puțin caracterele: `NULL`, `"`, `/`, `\`, `*`, `?`, `<`, `>`, `|`, `:` (utilizate în general pentru adresarea fișierelor)
ext3	da	255 octeți	nedefinit	orice byte mai puțin `NULL` poate fi folosit în numele fișierului
ReiserFS	da	4,032 octeți / 255 caractere	nedefinit	orice octet mai puțin `NULL` poate fi folosit în numele fișierului
HFS+	nu[3]	255 caractere UTF-16	nedefinit	orice caracter Unicode este permis
ISO9660	nu	`8.3` sau Joliet: 128 octeți	adâncime maximă a căii: 8 niveluri	există mai multe niveluri pentru standard, fiecare cu restricțiile lui
UDF	da	255 octeți	1023 octeți	orice octet mai putin NULL poate fi folosit în numele fișierului

Pe lângă aceste acțiuni, folosind programe speciale, se pot efectua și alte operații asupra unui sistem de fișiere, precum:

- redimensionarea unei partiții – duce implicit la o redimensionare a sistemului de fișiere (în cazul în care acest lucru nu se întâmpla, trebuie reformatată partiția);
- convertirea unui sistem de fișiere dintr-un tip în altul;
- mutarea unei partiții – de obicei nu are efecte asupra sistemului de fișiere aflat pe aceasta.

4.8.1 Crearea unui sistem de fișiere

După cum s-a menționat anterior, majoritatea sistemelor de fișiere au un suport fizic asociat unde se regăsesc efectiv datele. Suportul unui sistem de fișiere se numește

partiție. O partiție poate avea un singur sistem de fișiere. Partiționarea a fost descrisă în secțiunea 2.2.2.

Sistemul de fișiere este plasat pe o partiție prin formatare. Formatarea este procesul prin care pe partiție sunt scrise structuri de date specifice sistemului de fișiere. Aceste structuri sunt utilizate de driver-ul asociat sistemului de fișiere pentru a identifica și utiliza datelor scrise pe partiție.

Într-un sistem Linux, formatarea se realizează prin intermediul utilitarului **`mkfs`** cu sintaxa:

```
1 root@ubuntu:~# mkfs -t \textless{}tip_sistem_de_fisiere> \textless{}
  dispozitiv>
```

Exemplu de utilizare:

```
1 root@ubuntu:~# mkfs -t ext3 /dev/sda2
```

Comanda prezentata anterior formatează partiția a 2-a de pe primul disc utilizând sistemul de fișiere ext3. Pentru fiecare tip de sistem de fișiere există un program pentru formatare. Aceste programe au numele de forma **`mkfs.<fstype>`** și pot fi apelate direct pentru a formata o partiție. Comanda echivalentă exemplului anterior este următoarea:

```
1 root@ubuntu:~# mkfs.ext3 /dev/sda2
```

O operație suplimentară formatării este verificarea sectoarelor din cadrul partiției. Dacă un sector este defect, se va marca și nu va fi folosit pentru stocarea datelor. Următoarea comandă va determina verificarea sectoarelor și formatarea partiției `/dev/sda2`:

```
1 root@ubuntu:~# mkfs.ext3 -c /dev/sda2
```

Pentru a crea un sistem de fișiere MS-DOS pe o anumită partiție se poate folosi comanda **`mkdosfs`** care este un front-end pentru comenzile **`mkfs.vfat`** și **`mkfs.msdos`**. Cu optiunea `-F` se poate specifica tipul tabelei FAT care poate să fie pe 12, 16 sau 32 biți. Următoarea comandă va crea un sistem de fișiere FAT 32 pe partiția specificată:

```
1 root@ubuntu:~# mkdosfs -F 32 /dev/sda2
```

4.8.2 Montarea unui sistem de fișiere

Sistemele de operare oferă acces uniform la datele din toate tipurile de sisteme de fișiere prin intermediul ierarhiei de fișiere.

> Pentru ca un sistem de fișiere să fie vizibil sistemului de operare el trebuie întâi să fie montat (*mount*) într-un director, numit director de montare (*mount point*). Operația se numește **montarea unui sistem de fișiere**.

Această operație este executată de cele mai multe ori în mod automat de sistemul de operare:

- în Windows, sistemele de fișiere primesc automat litere diferite; pot fi ulterior modificate, cu anumite restricții, folosind `Control Panel > Administrative Tools > Computer Management > Storage > Disk Management`;

- în Linux, sistemele de fișiere sunt montate atât în locuri bine definite (definite de utilizator la instalare) cât și în locuri special create (de exemplu `/media/<nume>/`) pentru a găzdui sistemele de fișiere de pe medii externe, precum USB stick-uri, CD/DVD-ROM-uri etc.

În Mac OS X, sistemele de fișiere sunt montate automat în `/Volumes/<nume>`.

Montarea manuală

Pentru a monta un sistem de fișiere, într-un mediu Linux se utilizează comanda **mount**, cu sintaxa:

```
root@ubuntu:~# mount -t <tip_sistem_fisiere> [-o <optiuni>] <dispozitiv>
<director_montare>
```

În exemplul de mai jos se folosește **mount** pentru a monta partiția a 2-a de pe primul disc, cu sistemul de fișiere ext3, în directorul `/mnt/data` (directorul `/mnt/data` trebuie să existe anterior execuției comenzii).

```
root@ubuntu:~# mount -t ext3 /dev/sda2 /mnt/data
```

Montarea unui sistem de fișiere doar cu drepturi de citire (*read-only*) se poate realiza prin opțiunea `-r`:

```
root@ubuntu:~# mount -r -t ext3 /dev/sda2 /mnt/data
```

Alte exemple de utilizare a opțiunilor de montare se găsesc în secțiunea 4.10.

Montarea automată

Într-un sistem Linux, fișierul `/etc/fstab` descrie sistemele de fișiere care vor fi montate automat la pornirea sistemului de operare. Fișierul conține câte o linie pentru fiecare sistem de fișiere. Comentariile încep cu `#`.

Sintaxa unei linii este:

```
<dispozitiv> <director montare> <tip sistem fisiere> <optiuni> <dump> <
pas>
```

Câmpurile au următoarele semnificații:

- `<dispozitiv>` – fișier care indica locul (de obicei un dispozitiv) unde se găsește sistemul de fișiere;
- `<director montare>` – director existent în structura de directoare unde se va putea accesa conținutul sistemului de fișiere montat;
- `<tip sistem fișiere>` – tipul sistemului de fișiere;
- `<opțiuni>` – opțiuni pentru montarea sistemului de fișiere - diferă de la un sistem de fișiere la altul;
- `<dump>` – indică dacă să se realizeze automat backup al partiției utilizând programul **dump**; opțiunea este în prezent rar folosită, preferându-se alte metode de backup; acest parametru are în mod obișnuit are valoarea 0;

- <pas> – indică ordinea în care fsck va verifica sistemele de fișiere (0 = **fsck** nu va verifica sistemul de fișiere).

Un exemplu de linie din /etc/fstab este:

```
1 /dev/sda1               /               reiserfs defaults 0 0
```

Linia descrie montarea primei partiții de pe primul disc în /, utilizând sistemul de fișiere ReiserFS cu opțiuni implicite, fără backup și fără verificări.

După efectuarea unei modificări în /etc/fstab, comanda

```
1 root@ubuntu:~# mount -a
```

va monta toate partițiile menționate.

Fără niciun argument, **mount** afișează lista sistemelor de fișiere montate la un moment dat.

```
1 ubuntu@ubuntu:~$ mount
2 /dev/sda1 on / type reiserfs (rw)
3 none on /proc type proc (rw)
4 none on /sys type sysfs (rw)
5 usbfs on /proc/bus/usb type usbfs (rw)
6 none on /dev/pts type devpts (rw)
7 none on /dev/shm type tmpfs (rw)
8 tmpfs on /tmp type tmpfs (rw)
```

Se observă că în listă nu apar doar sistemele de fișiere montate manual sau cele montate automat (descrise în /etc/fstab). Există și sisteme de fișiere care sunt create și montate automat de sistem în ierarhia de fișiere pentru a oferi informații despre starea sistemului (precum /proc – vezi secțiunea 5.1.3).

Lista sistemelor de fișiere montate la un moment dat se poate regăsi de asemenea și în fișierul /etc/mtab.

Operația inversă montării unui sistem de fișiere se numește **demontare**. Pentru a realiza demontarea se utilizează comanda **umount**, cu două sintaxe posibile:

```
1 root@ubuntu:~# umount <dispozitiv>
2
3 root@ubuntu:~# umount <director_de_montare>
```

Folosind **mount** și **grep** se află tipul partiției /dev/sda2:

```
1 root@ubuntu:~# mount | grep /dev/sda2
2 /dev/sda2 on /home type reiserfs (rw)
```

Pentru a demonta acest sistem de fișiere se poate utiliza oricare din următoarele două comenzi:

```
1 root@ubuntu:~# umount /dev/sda2
2
3 root@ubuntu:~# umount /home
```

4.8.3 Repararea unui sistem de fișiere

După cum s-a menționat și la începutul acestui capitol, sistemele de fișiere moderne au suport pentru jurnalizare. Acest suport este util în mod deosebit atunci când se produce o cădere a sistemului, precum atunci când se întrerupe alimentarea cu energie electrică.

La repornire, sistemul de operare efectuează o verificare a integrității sistemelor de fișiere. La această verificare, se execută comanda **`fsck`** pentru fiecare sistem de fișiere. Aceasta comanda este particularizată pentru fiecare sistem de fișiere cu numele **`fsck.<tip_sistem_de_fisiere>`**, precum comanda **`mkfs`**.

Pentru a verifica un sistem de fișiere comanda **`fsck`** se apelează cu următoarea sintaxă:

```
1 root@ubuntu:~# fsck -t <tip_sistem_fisiere> <dispozitiv>
```

Dacă se dorește verificarea celei de-a doua partiții de pe primul disc, cunoscându-se că ar trebui să se găsească un sistem de fișiere ext3 pe aceasta, se utilizează comanda:

```
1 root@ubuntu:~# fsck -t ext3 /dev/sda2
```

În funcție de opțiunile care se folosesc la execuția comenzii, **`fsck`** poate repara erorile pe care le întâlnește.

4.8.4 Crearea unei imagini pentru un sistem de fișiere

Imaginea unui sistem de fișiere reprezintă copia fidelă într-un fișier a fiecărui octet care descrie sistemul de fișiere (inclusiv datele din acesta). Această imagine poate fi ulterior copiată pe o altă partiție sau arhivată pentru o eventuală recuperare a datelor dacă partiția sursă este distrusă accidental. O utilizare foarte frecventă a imaginilor se întâlnește în copierea conținutul CD/DVD-urilor pe alte medii de stocare.

Pentru a realiza o imagine a unui sistem de fișiere se utilizează comanda **`dd`** (vezi secțiunea 7.6.5. În exemplul de mai jos se copiază conținutul primei partiții (sistemul de fișiere) de pe discul al 2-lea în fișierul `/mnt/data/partitie1.img`.

```
1 root@ubuntu:~# dd if=/dev/sdb1 of=/mnt/data/partitie1.img
```

Această comandă este independentă de sistemul de fișiere căruia i se face imaginea. Lucrează direct la nivelul octeților care alcătuiesc sistemul de fișiere și păstrează întreaga structură internă a acestuia.

În cazul utilizării comenzii pe sisteme de fișiere foarte mari, se va observa că timpul de execuție este destul de mare. Motivul pentru care se întâmplă acest lucru este faptul că, în mod implicit, comanda **`dd`** lucrează cu blocuri de date de 512 octeți, ceea ce înseamnă ca pentru fiecare 512 octeți comanda execută o citire și o scriere. În prezent, buffer-ele discurilor au valori dimensiuni mult mai mari și pot lucra cu blocuri mai mari de 512 octeți.

Pentru a modifica dimensiunea blocului citit/scris de dd se utilizează parametrul `bs=<dimensiune>`, unde `<dimensiune>` poate să fie `xKB`, `xMB` etc.

Spre exemplu:

```
1 root@ubuntu:~# dd bs=2MB if=/dev/sdb1 of=/mnt/data/partitie1.img
```

va executa aceeași operație dar citind blocuri de 2MB.

Există și un dezavantaj al utilizării unor blocuri de date mai mari. În anumite condiții, nu se va putea face copierea unui bloc întreg, din diferite motive precum un defect hardware al suprafeței magnetice, sau ultimul bloc nu are dimensiunea de 2MB (deoarece dimensiunea totală nu este multiplu de 2MB). De cele mai multe ori se dorește recuperarea cât mai multor date de pe disc, de preferat toate. În aceste condiții trebuie determinat daca o viteza mai mare (dimensiune mai mare a blocului) este o decizie acceptabila știind ca se pot pierde date mai multe la citire.

Pentru recuperarea datelor de pe discuri cu defecte hardware, se poate folosi programul **ddrescue**[1]. **ddrescue** este o aplicație asemănătoare **dd** dar care este specializată în recuperarea cât mai multor date de pe discuri.

4.8.5 "Ştergerea" unui sistem de fișiere

Exista două tipuri de ștergere a unui sistem de fișiere:

- ștergerea logică
- ștergerea fizică

Ştergerea logică se obține prin ștergerea partiției corespunzătoare sistemului de fișiere din tabela de partiții (descris în secțiunea 2.2.2). Din moment ce nu mai există partiție care să încapsuleze sistemul de fișiere, acesta se poate considera șters. Practic, datele lui încă mai rezidă pe disc, până la o suprascriere cu alte date.

Faptul ca datele nu sunt șterse fizic de pe disc a pus și încă pune probleme oamenilor care doresc ca datele odată ce sunt șterse să rămână șterse pentru eternitate. Din acest motiv, pentru a elimina orice posibilitate de a recupera un sistem de fișiere sau datele dintr-un sistem de fișiere, se utilizează ștergerea fizică. Ştergerea fizică este realizată de fapt prin suprascrierea blocurilor de pe suportul fizic (unde era înainte sistemul de fișiere) cu date aleatoare sau cu zerouri.

Pentru suprascriere se utilizează cel mai rapid tot programul **dd** prezentat anterior dar, în loc ca partiția să fie folosită ca sursă, ea este folosită ca destinație pentru un set de date aleatoare (sau zerouri).

Pentru a suprascrie cu zerouri o partiție, se poate folosi comanda:

```
1 root@ubuntu:~# dd if=/dev/zero of=/dev/sda1
```

Pentru a suprascrie o partiție cu date aleatoare, se folosește comanda:

```
1 root@ubuntu:~# dd if=/dev/urandom of=/dev/sda1
```

4.8.6 Monitorizarea utilizării discului

O problemă care nu poate fi neglijată este umplerea hardisk-ului. Pentru a evita acest lucru, se folosesc comenzi care să afișeze cât spațiu este consumat, de exemplu comenzile **df** și **du**.

[1] http://www.gnu.org/software/ddrescue/ddrescue.html

Comanda **df** oferă informații despre dimensiunea totală, spațiul utilizat, spațiul liber și procentul de folosire al fiecărei partiții.

```
ubuntu@ubuntu:~$ df
Filesystem   1K-blocks      Used Available Use\% Mounted on
/dev/sda1     5859784  4449900   1409884 76\% /
/dev/sda2     15361340 10174596   5186744 67\% /home
```

Atunci când procentul de folosire depașește 80% se recomandă curățarea partiției.

Se poate folosi opțiunea `-a` pentru a include și pseudo sistemele de fișiere care sunt afișate cu dimensiunea 0, de exemplu `/proc`, `/sys`:

```
ubuntu@ubuntu:~$ df -a
Filesystem           1K-blocks      Used Available Use\% Mounted on
/dev/sda1                 5859784   4449900   1409884  76\% /
/dev/sda2                15361340  10174596   5186744  67\% /home
/proc                        0         0         0   -  /proc
/sys                         0         0         0   -  /sys
```

Comanda **du** caută în mod recursiv în directoarele specificate pentru a afișa cât spațiu consumă fiecare director. În exemplul de mai jos se afișează spațiul ocupat de directorul `/etc/network`:

```
ubuntu@ubuntu:~$ du /etc/network
8       /etc/network/if-down.d
12      /etc/network/if-post-down.d
12      /etc/network/if-pre-up.d
24      /etc/network/if-up.d
64      /etc/network
```

Pentru afișarea dimensiunii în format mai ușor de citit se poate folosi opțiunea `-h`:

```
ubuntu@ubuntu:~$ du -h /etc/network
8.0K    /etc/network/if-down.d
12K             /etc/network/if-post-down.d
12K             /etc/network/if-pre-up.d
24K             /etc/network/if-up.d
64K             /etc/network
```

Pentru a opri căutarea recursivă și a afișa doar spațiul ocupat de directorul transmis ca argument se folosește opțiunea `-s`:

```
ubuntu@ubuntu:~$ du -h -s /etc/network
64K             /etc/network
```

4.9 Tendințe în sistemele de fișiere

4.9.1 Sisteme de fișiere în userspace

Sistemele de fișiere pot fi citite/scrise de sistemele de operare doar dacă exista driver-e care să le implementeze. Implementarea unui driver pentru un sistem de fișiere nu este tocmai ușoară. Acesta este și principalul motiv pentru care exista destul de puține sisteme de fișiere care să implementeze un set de funcționalități acceptabil. Majoritatea sistemelor de fișiere prezente în sistemele de operare au mulți ani de dezvoltare.

După cum s-a prezentat anterior, există situații în care s-a putut foarte ușor abstractiza accesul la o serie de date prin intermediul fișierelor (precum informații despre sistem, în `/proc`). Practic, aproape orice informație poate fi prezentată sub forma de fișier. Singura problemaă care a împiedicat mult timp această trecere către reprezentarea de date folosind fișiere a fost dificultatea scrierii unui driver adecvat.

În ultima perioadă a luat amploare utilizarea de drivere pentru diferite sisteme de fișiere în user-space, spre deosebire de driverele clasice care funcționează în kernel-space. Driverele din userspace se bazează pe faptul că există un driver în kernel care se ocupă de translatarea apelurilor la date între kernel și sistemul de fișiere din userspace. Acest suport pentru sisteme de fișiere în userspace este prezent în kernel-ul Linux cu numele FUSE[1] (Filesystem în USErspace) iar în Mac OS X cu numele MacFUSE[2].

Tabelul 4.15 prezintă avantajele și dezavantajele sistemelor de fișiere în user-space:

Tabelul 4.15: Avantaje/dezavantaje sisteme de fișiere în user-space

Avantaje	**Dezavantaje**
suport pentru scrierea de driver-e în numeroase limbaje de programare; dezvoltare mult mai rapida datorită atât limbajelor de nivel înalt cât și datorită posibilității de testare mult mai rapidă (+ debugging); nu blochează sistemul în cazul unei proaste funcționări; compatibilitate a implementărilor între Linux și Mac OS X	mai lente decât sistemele de fișiere normale; au o flexibilitate mai mică în ceea ce privește nivelul de acces la hardware, comparativ cu un driver de kernel (de obicei nu este o limitare foarte importantă, dat fiind scopul sistemelor de fișiere de acest tip)

Printre cele mai folosite sisteme de fișiere în userspace se numără:

- NTFS-3G – sistem de fișiere care oferă suport read/write pentru NTFS;
- sshfs – sistem de fișiere care oferă acces la fișierele de pe altă mașină prin intermediul SSH ca și cum ar fi parte din sistemul de fișiere local;
- EncFS – Encrypted Filesystem – sistem de fișiere care criptează automat fișierele din directorul pe care îl folosește ca suport;
- AVFS – Anti-Virus Filesystem – sistem de fișiere care realizează scanarea pentru viruși în momentul în care se realizează un acces la fișiere; este implementat ca sistem de fișiere intermediar pentru accesul protejat la alt sistem de fișiere.

Fiecare dintre aceste sisteme de fișiere se montează într-un mod diferit, datorită modului în care au fost gândite să lucreze cu datele: unele au nevoie de un nume de utilizator și o parolă, altele au nevoie de calea către un alt sistem de fișiere etc. Mai multe informații se pot găsi pe paginile web ale fiecărui proiect.

[1] http://fuse.sourceforge.net/
[2] http://code.google.com/p/macfuse/

4.9.2 ZFS, ext4, btrfs

ZFS (*Zettabyte File System*) este un sistem de fișiere pe 128 de biți și logical volume manager care prezintă multe facilitați avansate cum ar fi clone copy-on-write, snapshot-uri, dynamic stripping, verificare continuă a integrității datelor, backup și restaurare rapidă, compresie incorporată, scalabilitate. ZFS a fost dezvoltat de Sun Microsystems ca software open-source și este sub licența CCDL.

ext4 este un sistem de fișiere jurnalizat dezvoltat ca îmbunătățire a lui ext3. ext4 aduce nou suport pentru sisteme de fișiere foarte mari (până la 1 exabyte = 1024 petabytes), extents (posibilitatea de a mapa blocuri mari contigue pentru un singur fișier), compatibilitate înapoi/înainte. Compatibilitatea înapoi este dată de faptul că un sistem ext3 poate fi montat ca un sistem ext4. Compatibilitatea înainte este posibilă deoarece un sistem ext4 poate fi montat ca un sistem ext3. ext4 a fost inclus în kernel-ul Linux începând cu versiunea 2.6.19 dar la momentul scrierii acestei cărți el este marcat ca fiind în dezvoltare.

btrfs (*Better File System*) este un sistem de fișiere de tipul copy-on-write dezvoltat pentru Linux. La fel ca și ext4, btrfs contine suport pentru extents. O caracteristică principală a acestui sistem de fișiere este posibilitatea efectuării unor operații critice în timpul funcționării, de exemplu defragmentarea, modificarea dimensiunii partiției și verificarea consistenței. btrfs prezintă multe alte caracteristici printre care efectuarea de snapshot-uri și de backup-uri incrementale. Inițial dezvoltat de către Oracle, btrfs este licențiat GPL.

4.10 Studii de caz

4.10.1 Comenzi pentru lucrul cu fișiere în Windows

Tabelul 4.16 prezintă echivalența comenzilor de bază Windows și Linux pentru prelucrarea fișierelor.

4.10.2 Utilizarea în siguranță a comenzilor pe fișiere

Exista o serie de opțiuni pentru comenzile **`cp`**, **`mv`** și **`rm`** care fac utilizarea acestora să fie mult mai sigură din punct de vedere al pierderii datelor în mod accidental.

Fiecare comandă are printre opțiuni `-i`, o opțiune care determină comanda să ceară o confirmare pentru fiecare fișier șters/suprascris. Pentru a anula efectul acestui parametru poate fi folosit parametrul `-f`.

Fiecare comandă recunoaște opțiunea `-v`, opțiune care determină afișarea fișierelor procesate la momentul respectiv. Această opțiune poate reprezenta o metodă bună pentru a vizualiza ce anume se întâmplă în spatele comenzii, mai ales dacă se execută comanda pentru un set mare de fișiere.

Aceste două opțiuni (`-iv`) pot fi forțate odată cu toate execuțiile realizate de utilizator pentru aceste comenzi dacă se realizează alias-uri. Alias-urile sunt descrise de obicei

Tabelul 4.16: Comenzi pentru lucrul cu fișiere în Windows

Comanda Linux	Comanda Windows	Descriere
`comanda --help`	`comanda /?`	afișează informații despre comandă
`cd`	`cd`	schimbă directorul curent
`pwd`	`chdir`	afișează directorul curent
`clear`	`cls`	șterge ecranul consolei curente
`cp`	`copy`	copiază un fișier
`rm`	`del`	șterge un fișier
`ls`	`dir`	afișează conținutul directorului curent
`vim`	`edit`	editează un fișier text
`exit`	`exit`	închide shell-ul curent
`diff`	`fc`	compară două fișiere și afișează diferențele între ele
`find`	`find`	caută fișiere
`mkfs (mke2fs)`	`format`	formatează un disc
`free`	`mem`	afișează informații despre memoria liberă și cea ocupată
`mkdir`	`mkdir`	creează un nou director
`mv`	`move`	mută un fișier
`mv`	`ren`	redenumește un fișier
`date`	`time`	afișează ora sistemului

într-un fișier care este încărcat la pornirea shell-ului (precum `~/.bashrc`) (vezi secțiunea 12.10):

```
1 alias cp="cp -iv"
2 alias mv="mv -iv"
3 alias rm="rm -iv"
```

Astfel, la execuția oricărei dintre aceste comenzi, automat se vor lua în considerare și aceste opțiuni.

4.10.3 Montarea unui sistem de fișiere FAT32 astfel încât toți utilizatorii să aibă drept de scriere pe el

Pentru a realiza acest lucru se utilizează opțiunile de montare a sistemelor de fișierelor. Pentru a forța la montare drepturile fișierelor se utilizează măști (*mask*). Masca este o înșiruire de biți care dacă sunt 1 semnifică un drept care lipsește.

Exista mai multe tipuri de măști:

- `umask` – mască aplicată tuturor fișierelor noi (vezi secțiunea 10.2.4);
- `fmask` – mască aplicată tuturor fișierelor;
- `dmask` – mască aplicată tuturor directoarelor;

Folosind aceste măști se definesc opțiuni pentru montarea unei partiții FAT32 în felul următor:

```
1 root@ubuntu:~# mount -t vfat -o dmask=000,fmask=111 /dev/sda2 /mnt/media
```

Parametrul `-o dmask=000,fmask=111` specifică faptul că pentru toți utilizatorii fișierele normale trebuie să apară cu drepturi read/write iar directoarele cu toate drepturile. Trebuie menționat că pe o partiție FAT32 nu ar trebui să se găsească aplicații/script-uri ce pot fi executate din Linux. Pentru a preveni această situație, tuturor fișierelor le este îndepărtat dreptul de execuție, prin masca `111`.

Comanda de mai sus se poate transforma foarte ușor într-o linie în `/etc/fstab`:

```
1 /dev/sda2  /mnt/media   vfat  defaults,dmask=000,fmask=111   0   0
```

S-a păstrat opțiunea `defaults` pentru că aceasta activează alte opțiuni utile la montarea automată.

4.10.4 Montarea unui sistem de fișiere FAT32 astfel încât utilizatorii dintr-un anumit grup să aibă drept de scriere pe acesta

Exista situații când se dorește ca doar un grup de utilizatori să aibă acces la o partiție. În aceste condiții, posesorul fișierelor trebuie modificat în `root` iar grupul în grupul dorit. Grupul trebuie să aibă toate drepturile. În primul rând trebuie determinat identificatorul grupului (`gid`, *group id*). care se dorește a avea drepturi asupra fișierelor. Cunoscându-se numele grupului, comanda următoare va afișa identificatorul grupului.

```
1 ubuntu@ubuntu:~$ cat /etc/group | grep numegrupnumegrup:x:100:
2
```

Comanda filtrează liniile din `/etc/group` și afișează pe ecran doar liniile care încep cu numele grupului dorit. Numărul ce se găsește între caracterele `:` este identificatorul grupului.

Comanda care montează un sistem de fișiere FAT32 cu restricțiile menționate mai sus este următoarea:

```
1 root@ubuntu:~# mount -t vfat -o dmask=007,fmask=117,gid=100,uid=0 /dev/
  sda2 /mnt/media
```

Se observă că, spre deosebire de cazul anterior, au mai fost introduse două opțiuni:

- `gid=100` – identificatorul grupului care va avea drepturi asupra fișierelor;
- `uid=0` – identificatorul utilizatorului; utilizatorul root are `uid=0`.

Măștile au fost schimbate pentru a nu permite niciun fel de drept altor utilizatori în afara grupului și utilizatorului `root` (au `7` pe ultima poziție).

4.10.5 Sistem de fișiere într-un fișier

După cum s-a prezentat și în subcapitolele anterioare, în Linux se poate accesa (aproape) tot printr-un fișier. În particular, fiecare dispozitiv are asociat un fișier. Spre

exemplu, prima partiție de pe primul HDD are asociat (de obicei) fișierul `/dev/sda1`. În mod similar cu utilizarea fișierului `/dev/sda1`, orice alt fișier poate fi folosit ca suport pentru un sistem de fișiere. Două dintre motivele cele mai folosite pentru a realiza acest lucru sunt:

- experimentarea unor noi sisteme de fișiere, fără a repartiționa discul;
- încapsularea datelor într-un sistem de fișiere (eventual criptat).

În tabelul 4.17 este prezentat modul în care se poate realiza un sistem de fișiere stocat într-un fișier. Sunt prezentate comenzile atât pentru sistemul de fișiere aflat într-o partiție cât și pentru sistemul de fișiere aflat într-un fișier.

Tabelul 4.17: Sisteme de fișiere peste partiții sau peste fișiere

Acțiune	**Sistem de fișiere**	
	într-o partitie	**într-un fișier**
alocare spațiu	se partiționează discul; se obține partiția `/dev/sda1`	`dd if=/dev/zero of=~/date bs=1MB count=20` se crează astfel un fișier cu 20 blocuri, fiecare 1MB, se obține echivalentul unei partiții de 20MB
formatare	`/sbin/mkfs -t vfat /dev/sda1`	`/sbin/mkfs -t vfat ~/date`
verificare	–	`file ~/date`
montare	`mount -t vfat /dev/sda1 dir/`	`mount -t vfat -o loop ~/date dir/`
demontare	`umount dir/`	`umount dir/`

Opțiunea `-o loop` specifică sistemului de operare că partiția nu se găsește pe un dispozitiv fizic ci într-un fișier.

4.10.6 ntfs-3g

nfts-3g[1] este proiectul care oferă suport read-write pentru partiții NTFS sub Linux. Pentru instalarea ntfs-3g se folosește comanda:

```
1 root@ubuntu:~# apt-get install ntfs-3g
```

Pentru a identifica partiția NTFS se poate utiliza comanda:

```
1 root@ubuntu:~# fdisk -l | grep NTFS
```

care afișează doar liniile care conțin `NTFS` din ieșirea comenzii **`fdisk -l`**,

Pentru a monta sistemul de fișiere în mod automat, în `/etc/fstab` se adauga o linie de forma:

```
1 <dispozitiv> <director montare> ntfs-3g defaults,locale=en_US.utf8 0 0
```

[1] http://www.ntfs-3g.org/

Cuvinte cheie

- sistem de fișiere
- montare
- demontare
- fsck
- fișier
- director
- jurnalizare
- cale absolută, cale relativă
- ierarhie
- copiere, mutare, ștergere, redenumire
- arhivare
- dezarhivare
- backup
- descriptor de fișier
- redirectare
- drept de acces
- ls, pwd, cd
- cp, mv, rm
- mkdir, rmdir, touch

Întrebări

1. Sistemele de fișiere din prezent oferă un grad mai mare de siguranță împotriva căderilor de tensiune prin implementarea:
   - ❑ drepturilor de acces la nivel de utilizator și grup
   - ❑ drepturilor de acces prin liste de acces
   - ❑ suportului pentru legături hard
   - ❑ suportului pentru jurnalizare
2. Care este fișierul folosit pentru montarea automată a unui sistem de fișiere în Linux?
   - ❑ `/etc/resolv.conf`
   - ❑ `/etc/fsck`
   - ❑ `/mnt/hda5`
   - ❑ `/etc/fstab`
3. Un fișier are drepturile 653. Utilizatorul ce deține fișierul rulează comanda:

```
1 chmod a-x,u+w,g-r,o+r
```

   Care sunt noile drepturi ale fișierului?
   - ❑ 606
   - ❑ 762
   - ❑ 534
   - ❑ 451

4. Care din următoarele comenzi NU poate fi aplicată asupra unui director?
   - ❑ **`file`**
   - ❑ **`touch`**
   - ❑ **`rm -r`**
   - ❑ **`cat`**
5. Care din următoarele NU este o intrare validă în sistemul de fișiere?
   - ❑ fișier executabil
   - ❑ director al cărui nume începe cu .
   - ❑ legătură simbolică către /bin
   - ❑ interfață de rețea de looback
6. Care din următoarele sisteme de fișiere este un sistem nativ Linux?
   - ❑ `HPFS`
   - ❑ `FAT32`
   - ❑ `EXT3`
   - ❑ `UFS`
7. Un utilizator configurează un fișier cu drepturile 755. Ce utilizator are dreptul de execuție pe fișier?
   - ❑ root
   - ❑ utilizatorul care deține fișierul
   - ❑ niciun utilizator
   - ❑ toți utilizatorii
8. Ce reprezintă litera `v` din comanda de mai jos?

```
1 ubuntu@ubuntu:~$ tar czvf a.tgz dir1
```

   - ❑ crearea unei arhive
   - ❑ dezarhivarea unei arhive
   - ❑ crearea unei arhive GZIP
   - ❑ afișarea de mesaje despre arhivare
9. Care comandă este folosită pentru montarea unui sistem de fișiere în Linux?
   - ❑ mount
   - ❑ umount
   - ❑ mountfs
   - ❑ fsck

10. Pentru a putea utiliza un sistem de fișiere pe o partiție dată, prima acțiune absolut necesară care trebuie executată este:
    - ❑ ștergerea sistemului de fișiere existent anterior
    - ❑ montarea sistemului de fișiere de pe partiția respectivă
    - ❑ formatarea partiției cu sistemul de fișiere dorit
    - ❑ schimbarea drepturilor de acces pentru directorul în care este montată partiția

Capitolul 5

Procese

It's a well-known fact that computing devices such as the abacus were invented thousands of years ago. But it's not well known that the first use of a common computer protocol occurred in the Old Testament. This, of course, was when Moses aborted the Egyptians' process with a control-sea

Tom Galloway

Ce se învață din acest capitol?

- Definiția unui proces; diferența între un program și un proces
- Noțiunea de multitasking
- Ierarhia de procese într-un sistem de operare
- Comenzi de vizualizare a proceselor (ps, pstree, top)
- Daemoni; rularea proceselor în background/foreground
- Noțiunea de semnal; administrarea proceselor prin intermediul semnalelor
- Comunicarea între procese; operatorul | (pipe) de comunicare între procese
- Administrarea proceselor/serviciilor în Windows

5.1 Noțiuni introductive

În lumea sistemelor de operare Unix (și nu numai) există două concepte fundamentale: **fișierul** și **procesul**. Fișierul este folosit pentru a abstractiza informația și modul de stocare și utilizare a acesteia în sistemul de operare. Pe de altă parte, procesul este folosit pentru a abstractiza executarea sarcinilor sistemului de operare.

5.1.1 Ce este un proces?

> În definiția sa cea mai simplă, **un proces** este un program aflat în execuție. Dacă un program este considerat o entitate pasivă, un proces este considerat o entitate activă.

Definiția anterioară simplifică semnificația pe care o are un proces, încadrându-se în gradul de detaliu pe care îl atinge capitolul curent.

Procesul este unitatea de bază a sistemului de operare folosită pentru a îndeplini sarcinile indicate de utilizator. Orice solicitare a utilizatorului se traduce în crearea unui proces și rezolvarea acelei solicitări. Un proces este, așadar, **o entitate de execuție**.

5.1.2 Deosebirea dintre un proces și un program

Un **program** este doar un fișier executabil care se află pe disc. Se spune că un astfel de fișier este imaginea din care se va realiza un proces. Un program este, așadar, o entitate pasivă, care nu are asociate noțiuni de utilizare a memoriei și procesorului (elemente de bază într-un sistem de calcul). Fișierele `/bin/ls`, `/usr/bin/vi` sunt exemple de programe:

```
1  razvan@anaconda:~$ ls -l /bin/ls
2  -rwxr-xr-x 1 root root 96216 2008-06-27 03:31 /bin/ls
3
4  razvan@anaconda:~$ ls -l /usr/bin/vi
5  lrwxrwxrwx 1 root root 20 2009-02-27 15:43 /usr/bin/vi -> /etc/
   alternatives/vi
6
7  razvan@anaconda:~$ ls -l /etc/alternatives/vi
8  lrwxrwxrwx 1 root root 18 2009-04-24 09:59 /etc/alternatives/vi -> /usr/
   bin/vim.basic
9
10 razvan@anaconda:~$ ls -l /usr/bin/vim.basic
11 -rwxr-xr-x 1 root root 1644100 2009-03-19 17:32 /usr/bin/vim.basic
```

Un program conține, în cadrul fișierului executabil, două componente (secțiuni) principale:

- secțiunea de cod, ce conține instrucțiunile de rulare a programului pe procesor;
- secțiunea de date, ce conține informațiile utilizate la execuția programului;

Pe de altă parte, un **proces** pornește în momentul execuției unui program. În acel moment, sistemul de operare creează o structură (o entitate, o abstracție) denumită proces. Dincolo de informațiile conținute în *imagine* (fișierul executabil de pe disc), sistemul de operare asociază procesului o *zonă de memorie* unde își va menține datele și unde se va afla codul de executat. De asemenea, procesului i se asociază *timp de rulare* a codului pe procesor și *alte informații* utile pentru interacțiunea cu alte procese sau componente ale sistemului de operare. Toate aceste elemente formează împreună **contextul procesului**. Mai multe informații se găsesc în secțiunea secțiunea 14.7.1.

Exemple de procese sunt rularea comenzii **`ls`** sau rularea editorului **`vi`**. În momentul rulării comenzii **`ls`**, sistemul de operare folosește imaginea `/bin/ls` și creează un

proces. Acest proces este răspunzător pentru utilizarea resurselor sistemului (memorie, procesor) pentru a oferi rezultatul rulării comenzii (în cazul de față afișarea conținutului directorului curent):

```
razvan@anaconda:~$ ls
Desktop   bin    code          media     packages  public_html  websites
Download  bkup   extra-school  mystuff   people    school
Maildir   books  junk          official  projects  svn-repos

razvan@anaconda:~$ ls -l /bin/ls
-rwxr-xr-x 1 root root 96216 2008-06-27 03:31 /bin/ls

razvan@anaconda:~$ file /usr/bin/file
/usr/bin/file: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.15, stripped
```

În exemplul de mai sus, a doua comandă are ca argument executabilul folosit pentru crearea procesului. La fel și a treia comandă. Pot fi considerate exemple triviale de *self-reference*[1].

Se poate observa că un proces este o instanță de execuție a unui program. Un program poate avea însă mai multe instanțe de execuție asociate (instanțe care pot rula simultan sau succesiv). Există, așadar, o asociere many to one (mai multe la unul) între procese și programe.

5.1.3 Structura unui proces

Un proces este definit, la nivelul sistemului de operare, ca o **structură de date**. Un proces are proprietăți și dispune de resurse, cele mai importante fiind:

- **identificatorul unui proces**, numit și **PID** (*process id*). Este un număr întreg care identifică unic în cadrul sistemului de operare procesul respectv;
- **imaginea procesului**, reprezentată de programul din care ia naștere procesul. Imaginea procesului conține codul și datele necesare pentru execuție;
- **starea procesului**. Un proces se poate afla în starea de rulare (în momentul în care codul asociat se execută pe procesor) sau în așteptare (un alt proces se află în rulare);
- **zonele de memorie utilizate**, ce reprezintă regiunile de memorie folosite de proces pentru a menține codul și datele;
- **fișierele deschise**. Un proces menține o tabelă cu fișierele deschise. De obicei dimensiunea tabelei este limitată astfel încât și numărul de fișierele pe care le poate deschide un proces este limitat. (În Linux această limită este stabilită implicit la 1024).

[1] http://http://en.wikipedia.org/wiki/Indirect_self-reference

Vizualizarea structurii unui proces utilizând procfs

În Linux există mai multe metode de a obține informații despre structura unui proces. Una din aceste metode este analiza **procfs**. Acesta este un sistem de fișiere virtual care oferă acces la informații din sistemul de operare. Pentru a analiza un anumit proces va trebui accesat directorul `/proc/pid`. `pid` este numărul ce identifică procesul (*process id*). În exemplul de mai jos a fost accesat directorul procesului cu pid-ul 6740:

```
1 razvan@asgard:~$ cd /proc/6740/
2
3 razvan@asgard:/proc/6740$ ls
4 auxv     cwd      exe  maps  mounts      oom_adj    root   stat   status
   wchan
5 cmdline  environ  fd   mem   mountstats  oom_score  smaps  statm  task
```

Diversele intrări din acest director oferă informații legate de proces:

- pid-ul este dat de numele directorului: **6740**;
- imaginea procesului este dată de legătura simbolică **exe**:

```
1 razvan@asgard:/proc/6740$ ls -l exe
2 lrwxrwxrwx 1 razvan razvan 0 2009-07-19 19:38 exe -> /usr/bin/vim.
  gnome
```

  După cum se vede, procesul de față este o instanță a **vim** în mediu grafic (GNOME).

- linia de comandă utilizată pentru crearea procesului:

```
1 razvan@asgard:/proc/6740$ cat cmdline
2 vi 1.txt
```

  Se observă că s-a rulat comanda **vi 1.txt**.

- directorul curent – cel în care s-a rulat comanda – este dat de legătura simbolică **cwd**:

```
1 razvan@asgard:/proc/6740$ ls -l cwd
2 lrwxrwxrwx 1 razvan razvan 0 2009-07-19 19:38 cwd -> /home/razvan/
  tmp/fd_test
```

  Rezultă că directorul curent al comenzii este `/home/razvan/tmp/fd_test`.

- zonele de memorie ale procesului sunt date de fișierul maps:

```
1 razvan@asgard:/proc/6740$ cat maps
2 08048000-081ce000 r-xp 00000000 03:03 180465     /usr/bin/vim.gnome
3 081ce000-081db000 rw-p 00186000 03:03 180465     /usr/bin/vim.gnome
4 081db000-08287000 rw-p 081db000 00:00 0          [heap]
5 [...]
6 b7fdc000-b7fde000 rw-p 00014000 03:03 178741     /lib/ld-2.3.6.so
7 bfc7a000-bfc8f000 rw-p bfc7a000 00:00 0          [stack]
```

  Nu vom insista pe aceste regiuni întrucât depășesc aria de cuprindere a capitolului de față.

- fișierele deschise de proces pot fi vizualizate prin accesarea subdirectorului `fd`:

```
1  razvan@asgard:/proc/6740$ cd fd
2
3  razvan@asgard:/proc/6740/fd$ ls
4  0  1  2  3  5  7
5
6  razvan@asgard:/proc/6740/fd$ ls -l
7  total 6
8  lrwx------ 1 razvan razvan 64 2009-07-19 19:48 0 -> /dev/pts/3
9  lrwx------ 1 razvan razvan 64 2009-07-19 19:48 1 -> /dev/pts/3
10 [...]
11 lrwx------ 1 razvan razvan 64 2009-07-19 19:48 7 -> /home/razvan/tmp
   /fd_test/.1.txt.swp
```

Nu vom insista pe descrierea tuturor intrărilor, dar se poate observa că fișierul cu numărul 7 este o legătură simbolică către `/home/razvan/tmp/fd_test/.1.txt.swp`. Acest fișier este copia de lucru a editorului vim pentru fișierul `1.txt`.

Informații detaliate despre procfs sunt descrise în secțiunea 5.1.3. Sistemul de fișiere procfs este folosit drept suport de comenzile care oferă informații despre procese. Astfel de comenzi sunt **`ps`**, **`pstree`**, **`pgrep`** sau **`top`**, despre care vom discuta în secțiunile următoare.

5.1.4 Stările unui proces. Multiprogramare și multiprocesare

Într-un sistem de operare rulează, simultan, mai multe procese. Totuși un singur proces se poate executa la un moment dat pe un procesor *single-core* (sau, cu alte cuvinte, un procesor *single-core* poate executa codul asociat unui singur proces). Pe un sistem *multi-core*[1] vor putea rula mai multe procese în același timp, numărul acestora fiind egal cu numărul de nuclee de procesare ale sistemului hardware[2].

Orice proces are asociată o stare de activitate. În mod obișnuit există mai multe stări în care se poate găsi un proces, dar noi ne vom referi doar la două:

- **rulare** (*running*): procesul rulează în acel moment pe unul din procesoarele sistemului;
- **așteptare** (*waiting*): procesul așteaptă eliberarea procesorului pentru a putea rula pe acesta.

Se observă, așadar, că este nevoie de un algoritm care să ofere fiecărui proces ocazia de a fi executat pe unul din procesoarele sistemului.

> Mecanismul de alocare a proceselor pe procesoarele sistemului poartă numele de **planificare** (*scheduling*). Subsistemul nucleului care se ocupă de alocare se numește **planificator de procese** (*scheduler*).

Rolul planificatorului este de a alege un proces care să ruleze pe un procesor și, la îndeplinirea unei condiții date, să suspende execuția procesului pentru a îl înlocui cu un

[1] http://en.wikipedia.org/wiki/Multi-core_(computing)

[2] Sistemul hardware poate avea un procesor cu mai multe nuclee sau mai multe procesoare, fiecare cu mai multe nuclee

altul. Înlocuirea unui proces cu un alt proces se numește **schimbare de context** (*context switch*).

La începutul istoriei sistemelor de operare, dacă un utilizator dorea rularea unui program, el încărca pe un suport conținutul programului și aștepta rularea acestuia (în contextul unui proces). Când procesul era planificat pe procesor, acesta rula complet până la încheierea absolută a execuției. Abordarea prezentată are dezavantajul lipsei de interactivitate: dacă un proces rulează foarte mult sau dacă trebuie să comunice des cu dispozitive periferice lente, atunci alte procese care doresc să ruleze vor trebui să aștepte un timp îndelungat.

Următorul pas în evoluția sistemelor de operare a fost introducerea noțiunii de **multiprogramare**. Înainte de a intra în detalii, trebuie amintită diferența de viteza dintre procesor/memorie și dispozitivele periferice. Astfel, dacă un proces accesează hard disk-ul, va trebui să aștepte foarte mult timp pentru ca informația să fie disponibilă (numeric, diferența dintre viteza unui procesor și viteza de acces la datele de pe un hard disk este de ordinul milioanelor).

În abordarea multiprogramată, mai multe procese coexistă în cadrul unui sistem de operare. În momentul în care un proces accesează un dispozitiv periferic lent, va trebui să aștepte furnizarea informației dorite; drept urmare, procesul respectiv este suspendat temporar, și un alt proces este planificat în locul său pe procesor (*context switch*). În acest fel se mărește interactivitatea sistemului. Dacă un proces se blochează în comunicația cu un dispozitiv periferic lent, acesta nu mai irosește timpul procesorului în așteptare, ci este înlocuit cu un alt proces.

Următorul pas a fost apariția conceptului de **multiprocesare** sau **multitasking**. Acesta a venit ca răspuns la cerințele din ce în ce mai mari de interactivitate a sistemului de operare în raport cu utilizatorul. Dezavantajul multiprogramării este că un proces care rulează mult timp pe procesor fără a executa o acțiune blocantă nu va lăsa alte procese să ruleze. Acest lucru înseamnă pierderea interactivității[1]. Soluția este asocierea, cu fiecare proces, a unui timp limită de rulare pe procesor (o **cuantă de timp**). La încheierea cuantei de timp, procesul va fi automat suspendat temporar și înlocuit cu un altul. În abordarea multitasking, schimbarea de context se va produce când procesul care rulează pe procesor execută o acțiune blocantă sau când își încheie cuanta de timp.

Toate sistemele de operare moderne sunt sisteme multiprocesare (multitasking).

5.2 Ierarhia de procese în Unix. Vizualizarea proceselor sistemului

În sistemele Unix (ca atare și în Linux), procesele sunt organizate ierarhic. Astfel, se poate spune despre un proces că este procesul **părinte** al unui alt proces; viceversa, se spune că un proces este procesul **fiu** pentru un altul. Un lucru de reținut este faptul că un proces are un singur proces părinte, dar poate avea mai multe procese fiu.

Modelul ierarhiei de procese în Unix este prezentat în figura următoare:

[1] Denumirea tehnică este *starvation*

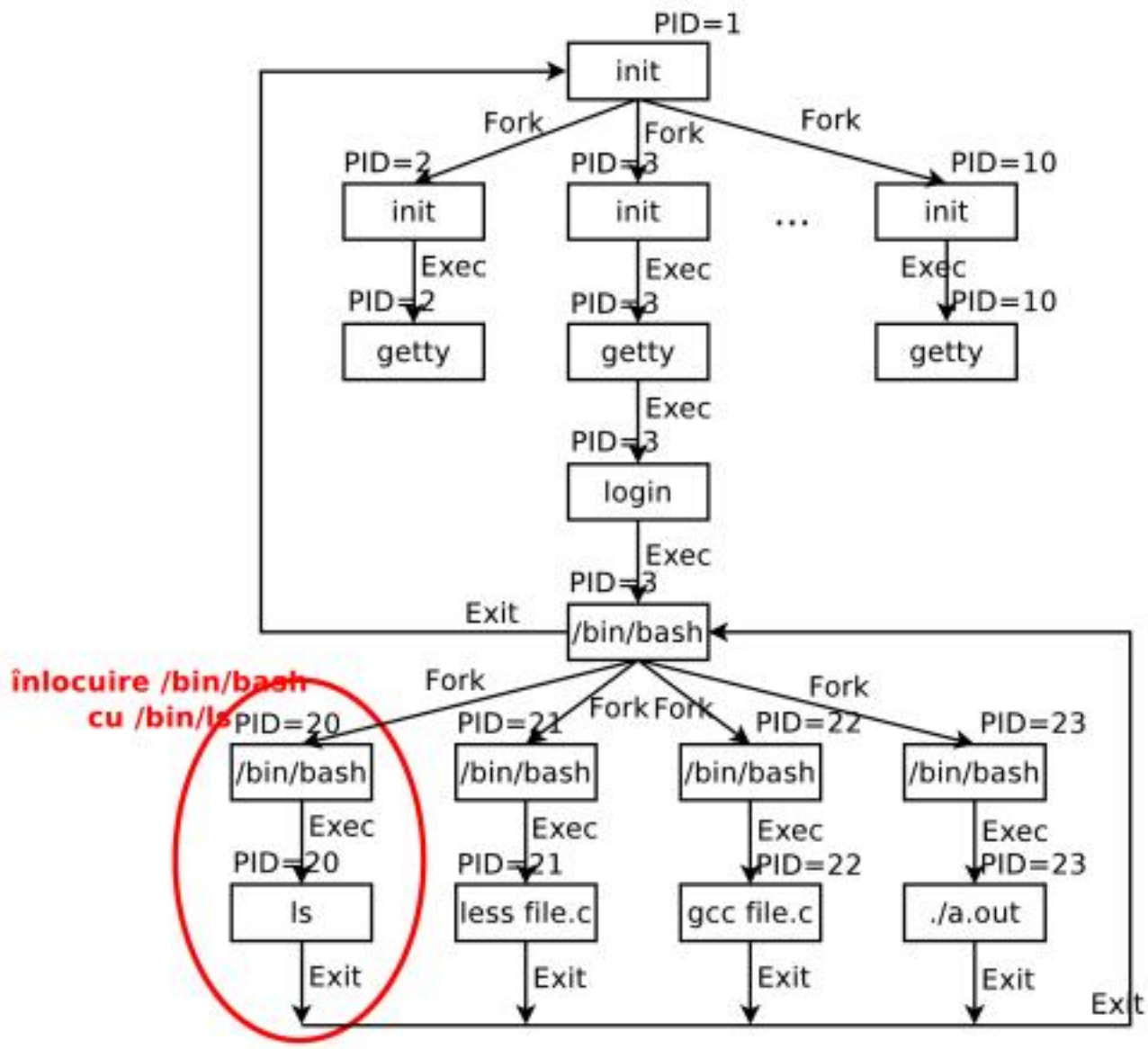


Figura 5.1: Ierarhia de procese în Unix

Ierarhia apare ca urmare a faptului că un proces este creat prin intermediul unui alt proces. La crearea ("nașterea") unui proces se realizează o copie a procesului părinte și se înlocuiește imaginea de program a părintelui cu cea dorită. Astfel, la rularea comenzii ls din shell (în imagine `/bin/bash`), se creează o copie a procesului shell, și se înlocuiește imaginea de executabil `/bin/bash` cu imaginea `/bin/ls`.

În Unix, părintele tuturor proceselor este denumit **init**. Imaginea de executabil asociată este `/sbin/init`. Acesta este primul proces creat de sistemul de operare. Este un proces special pentru că nu are părinte. Din init se creează alte procese, care la rândul lor vor crea alte procese, completându-se astfel ierarhia de procese ale sistemului.

5.2.1 Utilitarul ps

Comanda **ps** este principala comandă de vizualizare a proceselor sistemului. Informațiile prezentate de **ps** sunt obținute analizând informațiile din procfs. La o rulare simplă (fără argumente), **ps** afișează informații despre procesele pornite de utilizatorul curent pe terminalul activ:

```
1 razvan@asgard:~$ ps
2   PID TTY          TIME CMD
3  3036 pts/0    00:00:00 bash
4  3079 pts/0    00:00:00 ps
```

Rularea comenzii presupune afișarea unui cap de tabel și a unei intrări pentru fiecare proces. La o rulare simplă (fără argumente) se vor afișa identificatorul de proces (`PID`),

terminalul utilizat (`TTY`), timpul de rulare a procesului (`TIME`) și numele programului ce stă la baza acestuia (`CMD`). În exemplul de mai sus, utilizatorul a deschis un terminal cu un interpretor de comenzi (`bash`) și apoi a rulat comanda **`ps`**. Se observă că un proces **`ps`** se afișează și pe sine.

Comanda **`ps`** acceptă diferite tipuri de opțiuni care pot fi uneori conflictuale. În continuare vom insista pe opțiuni UNIX, caracterizate prin faptul că sunt precedate de semnul minus (`-`). Pentru detalii complete legate modul de utilizarea al comenzii și de efectul opțiunilor trebuie consultată pagina de manual asociată (**`man ps`**).

De exemplu, pentru afișarea proceselor pornite de utilizatorul curent, indiferent de terminalul utilizat, se folosește opțiunea `-a`.

```
1  razvan@asgard:~$ ps -a
2    PID TTY          TIME CMD
3   3074 pts/1    00:00:00 vi
4   3094 pts/0    00:00:00 ps
```

Se observă că, în această situație, se afișează procesele ce rulează pe terminalul `pts/1` și cele ce rulează pe `pts/0` fără a se mai preciza procesele tip shell de pe aceste terminale.

Comanda **`ps`** are un număr impresionant de argumente care pot controla conținutul și formatul afișării informațiilor despre procese. Vom prezenta cele două clase importante de opțiuni: opțiuni de selecție și opțiuni de formatare.

Opțiuni de selecție

Opțiunile de selecție presupun selectarea anumitor procese din sistem în funcție de diverse criterii. Sunt prezentate, în continuare, câteva opțiuni utile:

Selectarea tuturor proceselor din sistem: se realizează folosind argumentul `-e` sau `-A`:

```
1  razvan@asgard:~$ ps -e
2    PID TTY          TIME CMD
3      1 ?        00:00:00 init
4      2 ?        00:00:00 ksoftirqd/0
5      3 ?        00:00:00 events/0
6  [...]
7   3057 pts/1    00:00:00 bash
8   3074 pts/1    00:00:00 vi
9   3109 pts/2    00:00:00 bash
10  3131 pts/0    00:00:00 ps
```

Selecția proceselor după o anumită componentă: pid, comandă, terminal, utilizator; această selecție se realizează prin utilizarea unui argument specific (de exemplu `-C`, `-p` sau `-u`) urmat de lista de selecție (elementele listei sunt separate prin virgulă, fără spații între ele):

```
1  razvan@asgard:~$ ps -C bash,vi,gnome-terminal
2    PID TTY          TIME CMD
3   3034 ?        00:00:02 gnome-terminal
4   3036 pts/0    00:00:00 bash
5   3057 pts/1    00:00:00 bash
6   3074 pts/1    00:00:00 vi
```

```
 3109 pts/2    00:00:00 bash
 3180 pts/3    00:00:00 bash
```

În exemplul de mai sus au fost selectate procesele al căror executabil (`-C`) a fost `bash`, `vi` sau `gnome-terminal`.

```
razvan@asgard:~$ ps -p 1,3026,3074
  PID TTY          TIME CMD
    1 ?        00:00:00 init
 3026 ?        00:00:42 gedit
 3074 pts/1    00:00:00 vi
```

Cu comanda de mai sus s-au selectat procesele cu pid-ul (`-p`) 1, 3026 sau 3074.

```
razvan@asgard:~$ ps -u www-data
  PID TTY          TIME CMD
 2787 ?        00:00:00 apache2
 2788 ?        00:00:00 apache2
 2790 ?        00:00:00 apache2
```

În exemplul anterior s-au selectat procesele ce aparțin utilizatorului (`-u`) `www-data`.

Observați că deși s-a făcut selecția după utilizator, numele acestuia nu apare în rezultatul comenzii. Pentru aceasta vor trebui folosite opțiunile de formatare **`ps`** pentru afișarea altor câmpuri în afara celor implicite.

Opțiunile de selecție pot fi combinate. În exemplul de mai jos se selectează procesele `bash` sau `vi`, cele care au pid-ul 2968 sau 2972 și cele care aparțin utilizatorului `haldaemon`:

```
razvan@asgard:~$ ps -C bash,vi -p 2968,2972 -u haldaemon
  PID TTY          TIME CMD
 2583 ?        00:00:02 hald
 2590 ?        00:00:00 hald-addon-acpi
 2968 ?        00:00:04 metacity
 2972 ?        00:00:00 nautilus
 3036 pts/0    00:00:00 bash
 3057 pts/1    00:00:00 bash
 3074 pts/1    00:00:00 vi
 3109 pts/2    00:00:00 bash
 3180 pts/3    00:00:00 bash
```

Argumentele listei de selecție pentru opțiunile **`ps`** sunt separate prin virgulă, fără spații.

Se pot nega opțiunile de selecție cu ajutorul argumentului `-N`. Astfel, dacă se dorește selecția proceselor care nu aparțin utilizatorului privilegiat (`root`) și nici utilizatorului `razvan`, și care nu au ca executabil `apache2`, se va folosi comanda următoare:

```
razvan@asgard:~$ ps -N -u root,razvan -C apache2
  PID TTY          TIME CMD
 2283 ?        00:00:00 portmap
 2575 ?        00:00:00 dbus-daemon
 2583 ?        00:00:03 hald
 2590 ?        00:00:00 hald-addon-acpi
 2655 ?        00:00:00 exim4
 2663 ?        00:00:00 fbguard
 2664 ?        00:00:00 fbserver
 2741 ?        00:00:00 rpc.statd
 2756 ?        00:00:00 atd
```

Opțiuni de formatare

Opțiunile de formatare pentru comanda **ps** controlează coloanele care sunt afișate ca rezultat al rulării. Implicit, rularea comenzii oferă un cap de tabel cu elementele `PID`, `TTY`, `TIME` și `CMD`, urmat de o listă a proceselor. Aceste elemente pot fi schimbate prin intermediul opțiunilor de formatare.

Afișarea tuturor opțiunilor frecvente pentru **ps**:

```
1  razvan@asgard:~$ ps -F
2  UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
3  razvan    3036  3034  0  1574  3352   0 12:09 pts/0    00:00:00 bash
4  razvan    3248  3036  0   964   940   0 13:02 pts/0    00:00:00 ps -F
```

Se observă că s-a modificat capul de tabel. Sunt prezentate mai mult opțiuni utile, printre care `UID` (utilizatorul ce a creat procesul), `PPID` (pid-ul procesului părinte), `STIME` (timpul rulării procesului), `CMD` (comanda completă de rulare – cu argumente) etc.

În mod evident, opțiunile de formatare pot fi combinate cu opțiuni de selecție:

```
1  razvan@asgard:~$ ps -F -C bash,apache2,gnome-terminal,ps
2  UID        PID  PPID  C    SZ   RSS [...] TTY          TIME CMD
3  root      2786     1  0  2412  2632 [...] ?        00:00:00 /usr/sbin/
   apache2 -k start
4  www-data  2787  2786  0  2355  1948 [...] ?        00:00:00 /usr/sbin/
   apache2 -k start
5  www-data  2788  2786  0 57746  2452 [...] ?        00:00:00 /usr/sbin/
   apache2 -k start
6  www-data  2790  2786  0 57746  2456 [...] ?        00:00:00 /usr/sbin/
   apache2 -k start
7  razvan    3034     1  0  8424 15020 [...] ?        00:00:03 gnome-
   terminal
8  razvan    3036  3034  0  1574  3372 [...] pts/0    00:00:00 bash
9  razvan    3057  3034  0  1575  3348 [...] pts/1    00:00:00 bash
10 razvan    3109  3034  0  1575  3352 [...] pts/2    00:00:00 bash
11 razvan    3180  3034  0  1575  3340 [...] pts/3    00:00:00 bash
12 razvan    3257  3036  0   964   940 [...] pts/0    00:00:00 ps -F -C bash
   ,apache2,gnome-terminal,ps
```

În exemplul de mai sus, procesul cu pid-ul 2786 este procesul fiu al init (are PPID = 1). Următoarele trei procese sunt procesele fiu ale acestuia. De asemenea, procesul cu pid-ul 3034 (`gnome-terminal`) este tot un proces fiu al init, iar cele patru procese bash sunt procesele fiu ale acestuia. La fel, procesul ps (pid 3257) este procesul fiu al primului proces bash (pid 3036). O reprezentare ierarhică a celor zece procese de mai sus este următoarea:

```
1  init (1) ---> apache2 (2786)        ---> apache2 (2787)
2                                       \--> apache2 (2788)
3                                       \--> apache2 (2790)
4           \--> bash-terminal (3034) ---> bash (3036)      ---> ps (3257)
5                                       \--> bash (3057)
6                                       \--> bash (3109)
7                                       \--> bash (3180)
```

Afișarea de opțiuni specifice pentru procese

Specificarea opțiunilor de afișare pentru **ps** se realizează cu ajutorul argumentului `-o`, urmat de argumentele de formatare dorite (separate prin virgulă, fără spații).

Cele mai frecvent utilizate argumente de selecție sunt: `pid`, `ppid`, `args` (lista de argumente), `user` (utilizator), `comm` (comanda – doar numele executabilului), `cmd` (comanda cu argumente), `cputime` (timpul de procesor consumat), `group`, `pmem` (procent memorie ocupata), `state` (starea procesului – un singur caracter), `stat` (starea procesului – mai multe caractere), `tty` (terminalul de control asociat).

În continuare sunt prezentate câteva exemple.

Afișarea pid-ului (`pid`), utilizatorului (`user`), a stării procesului (`stat`), comanda cu argumente (`cmd`), timpul de procesor (`cputime`) și memoria folosită (`pmem`):

```
1  razvan@asgard:~$ ps -e -o pid,user,stat,cmd,cputime,pmem
2    PID USER     STAT CMD                              TIME %MEM
3      1 root     Ss   init [2]                      00:00:00  0.1
4      2 root     SN   [ksoftirqd/0]                 00:00:00  0.0
5      3 root     S<   [events/0]                    00:00:00  0.0
6  [...]
7   3009 razvan   S    /usr/lib/gnome-panel/notifi 00:00:00  1.4
8   3011 razvan   S    /usr/lib/gnome-applets/mixe 00:00:00  2.3
9   3016 razvan   Ss   gnome-screensaver           00:00:02  0.4
10 [...]
11  3180 razvan   Ss+  bash                        00:00:00  0.6
12  3260 razvan   R+   ps -e -o pid,user,stat,cmd, 00:00:00  0.1
```

Fără a insista pe semnificațiile stării proceselor, se observă că procesul `ps` (pid 3260) este în starea de rulare (R – running), iar celelalte sunt în starea de așteptare (S – sleeping). Sistemul de test dispune de un singur procesor și ca atare un singur proces poate rula la un moment dat.

Afișarea pid-ului, argumentelor, timpului de procesor, memoriei utilizate și timpului de pornire a comenzilor pentru procesele ce aparțin utilizatorului `www-data` și pentru procesele care au numele comenzii **bash**:

```
1  razvan@asgard:~$ ps -C bash -u www-data -o pid,args,cputime,rss,start
2    PID COMMAND                         TIME   RSS  STARTED
3   2787 /usr/sbin/apache2 -k start  00:00:00  1948 11:29:03
4   2788 /usr/sbin/apache2 -k start  00:00:00  2452 11:29:03
5   2790 /usr/sbin/apache2 -k start  00:00:00  2456 11:29:03
6   3036 bash                        00:00:00  3372 12:09:16
7   3057 bash                        00:00:00  3348 12:10:20
8   3109 bash                        00:00:00  3356 12:23:31
9   3180 bash                        00:00:00  3340 12:44:33
```

Spațiul de memorie utilizat (`rss` – resident set size), este măsurat în kiloocteți (KB).

Sortarea rezultatului rulării comenzii **ps** se realizează cu ajutorul opțiunii `--sort` urmată de lista de criterii de sortare. Semnul + (implicit) sau - în fața unui criteriu de sortare înseamnă sortare în ordine crescătoare sau descrescătoare:

```
1  razvan@asgard:~$ ps -e -o pid,cmd,cputime,pmem,rss --sort -rss
2   PID CMD                              TIME %MEM   RSS
3  3247 evince /home/razvan/school/ 00:00:37 10.7 206556
4   402 evolution                    00:01:28  7.5 146268
5  5121 /usr/bin/x-www-browser      00:01:03  5.5 107444
6  7709 pidgin                      00:00:05  2.9 56104
7  3334 /usr/bin/X :0 -audit 0 -aut 00:02:39  2.8 54960
```

```
4391 nautilus --no-default-windo 00:00:26  1.8 36324
4595 mono /usr/lib/tomboy/Tomboy 00:00:02  1.6 31252
 410 /usr/lib/evolution/evolutio 00:00:01  1.4 28844
4389 gnome-panel --sm-client-id  00:00:15  1.4 27616
[...]
```

Exemplul de mai sus realizează o sortare a proceselor din sistem după memoria ocupată. Se observă că procesele cele mai mari consumatoare de memorie sunt `evince` (un viewer de documente), `evolution` (un client de mail) și `x-www-browser` (un browser web).

Alte opțiuni

Utilitarul **`ps`** permite afișarea ierarhiei de procese în formă arborescentă. Acest lucru se realizează cu ajutorul opțiunii **`-H`**:

```
razvan@asgard:~$ ps -C init,apache2,gnome-terminal,bash -o pid,user,comm,
cputime,pmem --sort pid -H
  PID USER     COMMAND             TIME %MEM
    1 root     init            00:00:00  0.1
 2786 root       apache2       00:00:00  0.5
 2787 www-data     apache2     00:00:00  0.3
 2788 www-data     apache2     00:00:00  0.4
 2790 www-data     apache2     00:00:00  0.4
 3034 razvan     gnome-termina 00:00:05  2.9
 3036 razvan       bash        00:00:00  0.6
 3057 razvan       bash        00:00:00  0.6
 3109 razvan       bash        00:00:00  0.6
 3180 razvan       bash        00:00:00  0.6
```

În acest caz procesele fiu sunt indentate cu două caractere spațiu față de procesul părinte.

5.2.2 Utilitarul pstree

Utilitarul **`pstree`** afișează ierarhia de procese a sistemului. În cazul unei utilizări simple (fără niciun argument), comanda afișează ierarhia de procese începând de la init:

```
razvan@asgard:~$ pstree
init-+-acpid
     |-apache2-+-apache2
     |          `-2*[apache2---26*[{apache2}]]
     |-atd
     |-bonobo-activati
     |-clock-applet
     [...]
     |-gnome-terminal-+-bash
     |                |-bash---vi
     |                |-bash---man---pager
     |                |-bash---pstree
     |                |-gnome-pty-helpe
     |                `-{gnome-terminal}
[...]
```

Dacă este dorită afișarea ierarhiei de procese începând de la un anumit proces, va trebui transmis ca parametru pid-ul acelui proces:

```
razvan@asgard:~$ pstree 3034
gnome-terminal-+-bash
               |-bash---vi
               |-bash---man---pager
               |-bash---pstree
               |-gnome-pty-helpe
               `-{gnome-terminal}
```

Se poate afișa inclusiv pid-ul proceselor din ierarhie prin intermediul opțiunii `-p`:

```
razvan@asgard:~$ pstree -p 3034
gnome-terminal(3034)-+-bash(3036)
                     |-bash(3057)---vi(3074)
                     |-bash(3109)---man(3295)---pager(3302)
                     |-bash(3180)---pstree(3343)
                     |-gnome-pty-helpe(3035)
                     `-{gnome-terminal}(3037)
```

Ca și în cazul comenzii **ps**, **pstree** utilizează procfs pentru obținerea de informații despre ierarhia de procese.

5.2.3 Utilitarul pgrep

Comanda **pgrep** este echivalentul **grep** pentru lucrul cu procese și poate fi folosită pentru a afișa doar procesele din sistem care îndeplinesc o condiție. O funcționalitate echivalentă **pgrep** poate fi obținută cu ajutorul comenzii **ps** și al argumentelor specifice acesteia: `-p` pentru selecția după pid, `-C` pentru selecția după comandă, `-u` pentru selecția după utilizator etc.

În continuare sunt prezentate exemple ale rulării **pgrep** pentru selecția proceselor care au ca părinte procesul init (pid 1). Implicit se selectează doar pid-urile proceselor. Dacă se dorește și afișarea numelor proceselor, se folosește opțiunea `-l`:

```
razvan@asgard:~$ pgrep -P 1
2
3
4
5
1021
2284
[...]
3025
3027
3052

razvan@asgard:~$ pgrep -P 1 -l
2 ksoftirqd/0
3 events/0
4 khelper
5 kthread
1021 udevd
2284 portmap
[...]
3025 gnome-screensav
```

```
22 3027 gnome-terminal
23 3052 soffice
```

5.2.4 Utilitarul top

Utilitarul **top** este utilizat pentru a afișa informații în mod dinamic (în timp real) despre procesele existente în sistem. O utilizare frecventă a acesteia este monitorizarea sistemului (vezi secțiunea 10.3.1).

La o rulare fără argumente, **top** va afișa un ecran cu informații despre sistem și procesele care rulează. În mod implicit, acest ecran este actualizat la fiecare 3 secunde. Un exemplu de ecran afișat prin rularea comenzii **top** este prezentat mai jos:

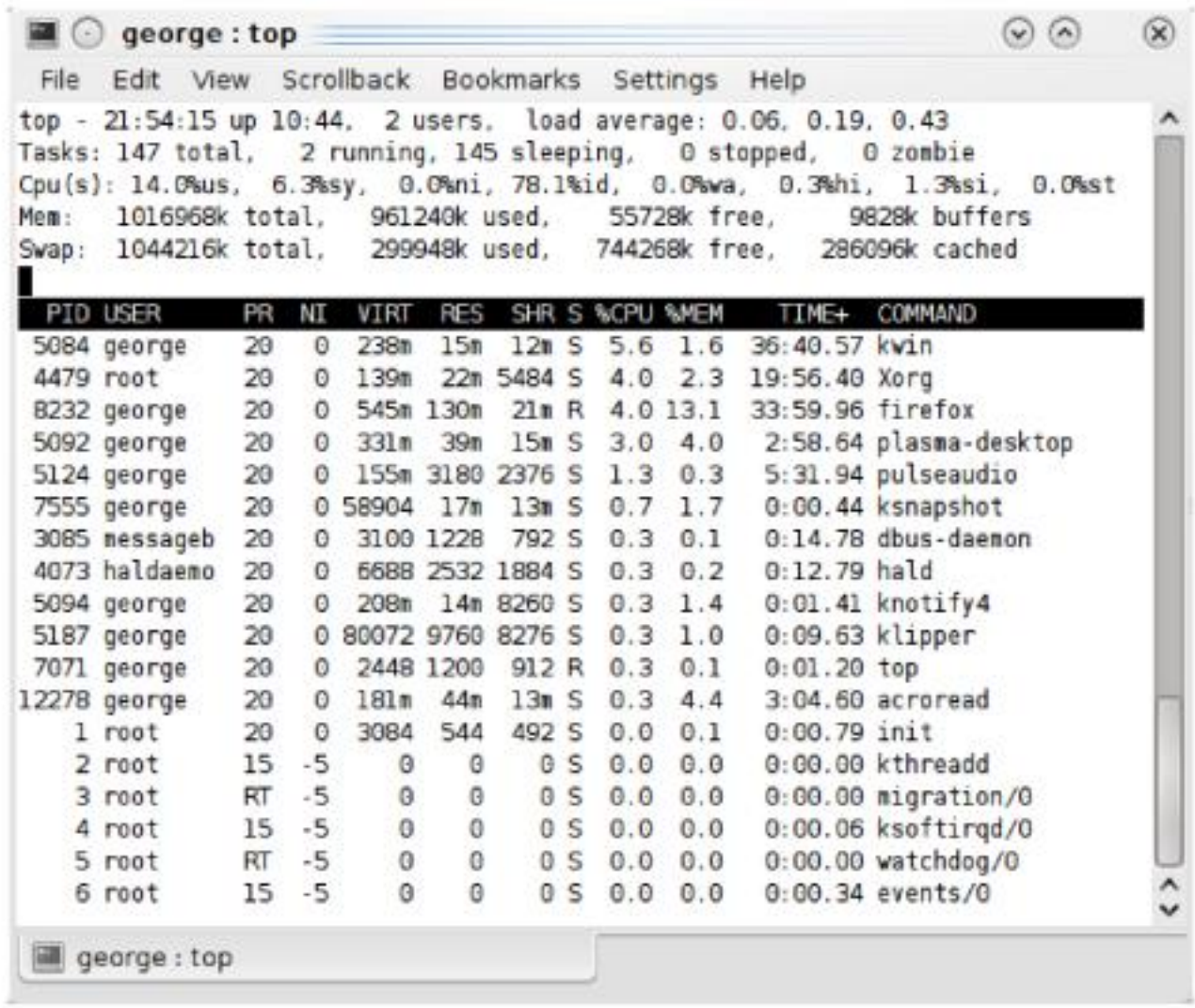


Figura 5.2: Ecran de rulare top

Ecranul afișat de top cuprinde următoarele zone:

- **zona de sumarizare** (*summary area*): aceasta conține informații generale despre sistem: încărcarea sistemului (prima linie), numărul de procese și starea lor (a doua linie), timpul de procesor utilizat (a treia linie), memoria și spațiul de swap ocupat (ultimele două linii). Gradul de încărcare a sistemului este o măsura a încărcării procesorului și poate fi determinat și cu ajutorul comenzii **uptime**.
- **zona de comandă** (*prompt line*): aici utilizatorul poate introduce comenzi specifice de interacțiune cu utilitarul **top**;
- **antetul de tabel** (*columns header*): afișează antetul de tabel pentru informațiile afișate despre procese;
- **zona de procese** (*task area*): afișează informații despre procese.

Interfața interactivă top

top are o interfața interactivă cu utilizatorul care îi permite acestuia alterarea formatului de afișare. Comanda cea mai utilă pentru lucrul interactiv cu **top** este cea de afișare a ecranului de ajutor (tasta `h`).

Majoritatea comenzilor top sunt de tipul *toggle*, adică afișarea sau oprirea afișării unei anumite componente din ecranul de afișare. În continuare vor fi prezentate câteva din comenzile top cele mai utile în funcție de zona ecranului de afișare în care au efect. Detalii suplimentare se pot afla prin consultarea paginii de ajutor (prin apăsarea tastei `h`) sau a paginii de manual (`man top`).

Comenzi pentru zona de sumarizare. Comenzile folosite pentru lucrul cu zona de sumarizare sunt comenzi de tipul *toggle*:

- tasta `l` este folosită pentru a activa/dezactiva afișarea de informații despre încărcarea sistemului (prima linie din zona de sumarizare);
- tasta `t` este folosită pentru a activa/dezactiva afișarea de informații despre numărul de procese și utilizarea procesorului (liniile 2 și 3 din zona de sumarizare);
- tasta `m` este folosită pentru a activa/dezactiva afișarea de informații despre utilizarea memoriei sistemului (ultimele două linii din zona de sumarizare).

Comenzi pentru zona de procese. Dintre comenzile care afectează zona de afișare pentru procese amintim:

- tasta `c` este folosită pentru a afișa comanda completă, nu doar numele procesului;
- tasta `f` este folosită pentru a adăuga/elimina coloane utilizate pentru afișarea de informații despre procese;
- tasta `o` este folosită pentru a schimba ordinea coloanelor din tabel; după apăsarea tastei `f` sau `o`, utilizatorului îi este prezentat un ecran de configurare, în care poate specifica noi coloane sau o nouă ordine (tot prin intermediul unor taste);
- tasta `R` (de tip *toggle*) este folosită pentru sortare ascendentă sau descendentă;
- tasta `F` sau tasta `O` sunt folosite pentru a configura coloana după care se face sortarea proceselor din zona de procese. În mod implicit sortarea se realizează după timpul de procesor, astfel încât procesele care într-un interval dat de timp au utilizat cel mai mult procesorul vor apărea primele în listă; tastele `<` și `>` pot fi folosite pentru schimbarea coloanei după care se relizează sortarea, selectând coloana din stânga sau din dreapta.

Comenzi în zona de comandă Aceste comenzi folosesc zona de comandă și afectează, de obicei, procesele sau zona de procese:

- tasta `k` permite terminarea unui proces. Apăsarea acestei taste oferă promptul `PID to kill:` unde utilizatorul va introduce pid-ul procesului a cărui execuție se dorește a fi încheiată;

- tasta `n` permite precizarea numărului de procese care vor fi afișate în zona de procese; apăsarea acestei taste conduce la apariția unui prompt unde utilizatorul precizează numărul de procese pe care dorește să le vizualizeze/monitorizeze;
- tasta `u` permite precizarea unui utilizator, fiind apoi afișate numai procesele care aparțin acestuia. Zona de comandă afișează promptul `Which user (blank for all)`: unde se cere introducerea unui nume de utilizator.

Interfața neinteractivă top

`top` poate fi utilizat și în mod neinteractiv prin folosirea argumentului `-b` în linie de comandă. Pornirea neinteractivă este folosită de obicei împreună cu argumentul `-n`, care precizează numărul de iterații de afișare. Se obține astfel un rezultat ce poate fi redirectat într-un fișier pentru analiză ulterioară:

```
1  george@asgard:~$ top -b -n 1
2  top - 23:03:58 up 11:54,  2 users,  load average: 0.11, 0.52, 0.52
3  Tasks: 148 total,   3 running, 145 sleeping,   0 stopped,   0 zombie
4  Cpu(s): 13.4%us,  6.3%sy,  2.0%ni, 77.1%id,  0.7%wa,  0.2%hi,  0.3%si,
   0.0%st
5  Mem:   1016968k total,   976164k used,    40804k free,    12684k buffers
6  Swap:  1044216k total,   395716k used,   648500k free,   371860k cached
7
8    PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
9   5124 george    20   0  155m 2656 2324 S  2.0  0.3   6:23.52 pulseaudio
10  8232 george    20   0  553m 134m  20m S  2.0 13.5  38:13.04 firefox
11 11446 george    20   0  2444 1096  824 R  2.0  0.1   0:00.01 top
12     1 root      20   0  3084  540  488 S  0.0  0.1   0:00.79 init
13     2 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kthreadd
14     3 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0
15     4 root      15  -5     0    0    0 S  0.0  0.0   0:00.07 ksoftirqd/0
16     5 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
17     6 root      15  -5     0    0    0 S  0.0  0.0   0:00.41 events/0
18     7 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 cpuset
19     8 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 khelper
20    11 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 netns
21 [...]
```

5.2.5 Timpul de execuție al unui proces. Comanda time

Timpul de execuție al unui proces se referă la durata sa de execuție în sistem, din momentul creării și până în momentul terminării acestuia.

Comanda **`time`** este utilizată pentru măsurarea timpului de rulare a unui proces și a resurselor utilizate de acesta. Rezultatul rulării implicite a comenzii **`time`** oferă 3 timpi:

- **real** – timpul efectiv de execuție, măsurat de la crearea procesului și până la terminarea acestuia;
- **user** – timpul petrecut în spațiul utilizator;
- **sys** – timpul petrecut rulând apeluri de sistem (în spațiul kernel).

time primește ca argument numele unui program a cărui durata de execuție va fi măsurată. Un exemplu de rulare este prezentat mai jos:

```
root@asgard:/home/razvan# time updatedb

real    4m19.309s
user    0m2.680s
sys     0m5.550s
```

Se observă că rularea comenzii **updatedb** a durat 4 minute și 19 secunde. Totuși timpul real de rulare a procesului se măsoară adunând timpul petrecut în spațiul utilizator (2.68 secunde) și timpul petrecut în spațiul kernel (5.55 secunde). Cu alte cuvinte, din timpul cât a durat rularea comenzii, procesul a consumat doar 8.23 secunde pentru a fi executat de către procesor. Restul timpului a fost ocupat cu rularea altor procese sau schimbarea contextului între procese. Schimbarea contextului are loc în momentul în care procesului îi expiră cuanta de timp sau accesează dispozitive periferice lente.

5.2.6 Sistemul de fișiere procfs

După cum s-a prezentat, sistemul de fișiere procfs este utilizat de comenzile ce obțin informații despre procesele din sistem: **ps**, **pstree**, **pgrep** sau **top**. Pachetul Debian/Ubuntu utilizat pentru instalarea acestor utilitare se numește `procps` pentru a indica faptul că procfs este utilizat pentru obținerea de informații despre procese.

procfs este un sistem de fișiere virtual (adică datele nu se găsesc pe un suport fizic permanent precum un hard disk sau CD-ROM). În cazul procfs, fiecare fișier este de fapt o regiune de memorie (stocată fizic în RAM) care oferă informații despre sistem. procfs oferă informații atât despre procesele din sistem cât și alte informații specifice: dispozitivele conectate, ocuparea întreruperilor etc, permițând totodată și configurarea anumitor componente ale sistemului.

procfs este montat în `/proc`:

```
razvan@asgard:~$ cat /etc/mtab
/dev/hda3 / ext3 rw,errors=remount-ro 0 0
tmpfs /lib/init/rw tmpfs rw,nosuid,mode=0755 0 0
proc /proc proc rw,noexec,nosuid,nodev 0 0
[...]
```

Din punctul de vedere al utilizatorului, procfs este vizibil ca o suită de fișiere și directoare cu informații despre sistem:

```
razvan@asgard:~$ cd /proc/

razvan@asgard:/proc$ ls
1/     2584/  2792/  2961/  3027/  97/          filesystems  mtrr
1021/  2585/  2793/  2963/  3028/  99/          fs/          net/
192/   2591/  2794/  2969/  3029/  acpi/        ide/         partitions
193/   2603/  2796/  2971/  3052/  asound/      interrupts   self
194/   2656/  2873/  2976/  3066/  buddyinfo    iomem        slabinfo
[...]
```

Informații despre procese

Informații despre procesele din sistem sunt oferite de procfs prin intermediul directoarelor al căror nume este un număr. Acest număr reprezintă pid-ul procesului despre care se dorește aflarea de informații.

În continuare vom prezenta diverse informații conținute în directorul din `/proc` asociat unui proces. Vom prezenta ca studiu de caz procesul folosit pentru editarea unui document Word. Fiind vorba de OpenOffice, numele executabilului din care a fost generat procesul este `soffice.bin`:

```
razvan@asgard:/proc$ ps -C soffice.bin
 PID TTY          TIME CMD
 3066 ?        00:01:02 soffice.bin

razvan@asgard:/proc$ cd 3066/

razvan@asgard:/proc/3066$ ls -F
auxv     environ  maps    mountstats  root    statm   wchan
cmdline  exe      mem     oom_adj     smaps   status
cwd      fd/      mounts  oom_score   stat    task/
```

Se poate observa că procesul asociat OpenOffice are pid-ul 3066 și ca atare va fi accesat directorul `3066`. Acest director conține o serie de fișiere, directoare și legături simbolice care oferă informații despre proces. Astfel, legătura simbolică `exe` oferă informații despre executabilul utilizat pentru crearea procesului; fișierul `cmdline` și fișierul `environ` precizează linia de comandă folosită și mediul de creare (environment):

```
razvan@asgard:/proc/3066$ ls -l exe
lrwxrwxrwx 1 razvan razvan 0 2007-08-04 12:01 exe -> /usr/lib/openoffice/
program/soffice.bin

razvan@asgard:/proc/3066$ cat cmdline
/usr/lib/openoffice/program/soffice.bin-writer-splash-pipe=5

razvan@asgard:/proc/3066$ cat environ
SSH_AGENT_PID=2950SHELL=/bin/bashGTK_RC_FILES=/etc/gtk/gtkrc:/home/razvan
/.gtkrc-1.2-gnome2OLDPWD=/usr/lib/openofficeUSER=
razvanOPENOFFICE_MOZILLA_FIVE_HOME=/usr/lib/openoffice/
programLD_LIBRARY_PATH=/usr/lib/openoffice/program
[...]
```

Fișierul `status` oferă, printre altele, informații referitoare la utilizatorul ce deține fișierul executabil (`uid 1000`), starea procesului (`S` – sleep), cantitatea de memorie ocupate, numărul de thread-uri etc:

```
razvan@asgard:/proc/3066$ cat status
Name:   soffice.bin
State:  S (sleeping)
SleepAVG:       98%
[...]
Uid:    1000    1000    1000    1000
Gid:    1000    1000    1000    1000
[...]
VmPeak:   205720 kB
VmSize:   202392 kB
VmLck:         0 kB
```

```
VmHWM:     84040 kB
VmRSS:     80968 kB
[...]
Threads:        6
[...]
```

Directorul `task/` oferă informații despre thread-urile acestui proces. Nu insistăm pe noțiunea de thread, deoarece depășește aria de cuprindere a acestui capitol.

```
razvan@asgard:/proc/3066$ cd task/

razvan@asgard:/proc/3066/task$ ls
3066  3067  3068  3069  3070  3071
```

Directorul `fd/` deține legături simbolice către fișiere deschise de procesul curent. Se observă, printre fișiere deschise, și fișierul cu extensia .doc care este editat (Raport.doc):

```
razvan@asgard:/proc/3066$ cd fd/

razvan@asgard:/proc/3066/fd$ ls
0  10  12  14  16  18  2   22  24  26  28  3   31  33  35  37  4  6  8
1  11  13  15  17  19  20  23  25  27  29  30  32  34  36  39  5  7  9

razvan@asgard:/proc/3066/fd$ ls -l
total 38
lr-x------ 1 razvan razvan 64 2009-08-04 13:31 0 -> /dev/null
l-wx------ 1 razvan razvan 64 2009-08-04 13:31 1 -> pipe:[8770]
l-wx------ 1 razvan razvan 64 2009-08-04 13:31 10 -> pipe:[9795]
[...]
lr-x------ 1 razvan razvan 64 2009-08-04 13:31 22 -> /dev/urandom
[...]
lrwx------ 1 razvan razvan 64 2009-08-04 13:31 34 -> /home/razvan/Raport.
doc
[...]
```

Alte informații utile

În afara informațiilor despre procese, procfs oferă și alte informații utile despre sistem. Acestea pot fi aflate tot prin accesarea unor fișiere și directoare din `/proc`. Printre informațiile utile (folosite, de asemenea, de diverse utilitare din sistem), se găsesc informații despre:

- procesor: prin accesarea `/proc/cpuinfo`;
- memoria sistemului: prin accesarea `/proc/meminfo`;
- partiții: prin accesarea `/proc/partitions`;
- timpul de rulare a sistemului: prin accesarea `/proc/uptime`;

În plus, accesarea directoarelor de forma `/proc/fs/`, `/proc/net/`, oferă informații suplimentare, permițând totodată configurarea unor parametri ai sistemului. Astfel procfs nu este numai o interfață de citire a informațiilor despre sistem, ci este și una de configurare a unor parametri ai sistemului.

5.3 Rularea proceselor în background. Job-uri. Daemoni

Rularea unui proces din interpretorul de comenzi (shell) rezultă de obicei în afișarea unui rezultat și terminarea procesului. Spre exemplu, dacă se dorește afișarea conținutului directorului curent se rulează comanda **`ls`**. Acest lucru duce la crearea unui proces fiu din shell, încărcarea executabilului `/bin/ls` și execuția noului proces încheiată cu terminarea acestuia. După terminarea procesului, utilizatorul poate rula un nou proces prin introducerea comenzii corespunzătoare la promptul interpretorului.

Un proces care rulează conform scenariului de mai sus se spune că rulează în **foreground** (în prim plan). Un proces care rulează în foreground are acces la terminalul curent. Altfel spus, procesele care rulează în foreground pot citi de la intrarea standard (standard input) și pot afișa informații la ieșirea standard (standard output).

De partea cealaltă, un proces poate rula în **background** (în fundal). Un proces care rulează în fundal pierde posibilitatea de a citi de la intrarea standard, dar își continuă rularea.

5.3.1 Rularea unui proces în background

Rularea unui proces în fundal se realizează cu ajutorul operatorului `&` după comandă. Acest operator va crea procesul asociat comenzii introduse și va forța rularea acestuia în fundal. Avantajul acestei abordări este faptul că se oferă înapoi promptul interpretorului către utilizator, în felul acesta utilizatorul putând introduce o nouă comandă. De obicei acest lucru se realizează când comanda de executat durează mult timp, ca de exemplu căutarea unui fișier, actualizarea bazei de date de căutare (**`updatedb`**), rularea unui proces cu interfață grafică etc. Procesele care rulează în fundal sunt denumite *job*-uri.

În exemplul de mai jos, se rulează comenzile de deschidere a editorului XEmacs și a calculatorului din mediul grafic GNOME. Ambele programe sunt pornite în fundal:

```
1 razvan@asgard:~$ xemacs &
2 [1] 3418
3
4 razvan@asgard:~$ gnome-calculator &
5 [2] 3420
6
7 razvan@asgard:~$
```

După rularea unei comenzi în fundal se observă că apare un mesaj specific de forma `[N] M`. În acest format `N` este indicele *job*-ului (indicele procesului care rulează în fundal), iar `M` este identificatorul de proces (pid-ul) pentru *job*-ul din fundal. Avantajul rulării acestor procese în fundal, așa cum a fost precizat și mai sus, este faptul că interpretorul de comenzi oferă înapoi promptul utilizatorului permițându-i astfel rularea de noi comenzi.

5.3.2 Suspendarea unui proces

Un proces care rulează în *foreground* poate fi forțat să intre în *background* (și implicit să intre în starea suspendat – nu rulează) prin intermediul combinației de taste `CTRL-Z` (control ținut apăsat, după care se apasă z). Apăsarea acestei combinații de taste este interceptată de sistemul de operare și procesul activ (aflat în *foreground*) este forțat să fie suspendat.

În exemplul de mai jos, utilizatorul a dorit editarea fișierului `out.txt` folosind *vi*. Procesul creat prin execuția comenzii **`vi out.txt`** rulează inițial în *foreground*. Utilizatorul a apăsat apoi `CTRL-Z` și a forțat suspendarea procesului curent în *background* și oferirea promptului către utilizator:

```
1  razvan@asgard:~$ vi out.txt
2  [3]+  Stopped                 vi out.txt
3
4  razvan@asgard:~$
```

Se observă că procesul a fost suspendat (*stopped*) și s-a creat jobul cu numărul 3.

Un scenariu asemănător este prezentat și în exemplul de mai jos. De data aceasta utilizatorul folosește utilitarul **`du`** pentru a vedea spațiul ocupat de fișierele și directoarele din directorul curent:

```
1  razvan@asgard:~$ du -hs *
2  1022M   Desktop
3  12K     Download
4  528K    bin
5  24M     books
6  756K    code
7  532K    junk
8  3.2M    official
9  4.0K    out.txt
10 29M     packages
11 165M    people
12 617M    pictures
13 116M    projects
14
15 [4]+  Stopped                 du -hs *
16
17 razvan@asgard:~$
```

5.3.3 Controlul job-urilor

Termenul de *job* a fost introdus referitor la procesele care rulează sau sunt suspendate în fundal. Interpretorul de comenzi (shell-ul) bash pune la dispoziția utilizatorului mai multe comenzi prin intermediul cărora se poate interacționa cu procesele din fundal. Mai multe informații despre aceste comenzi pot fi găsite prin consultarea paginii de manual (**`man bash`**, secțiunea `JOB CONTROL`) sau pagina info (**`info bash`**, secțiunea "JOB CONTROL". Trebuie instalat pachetul `bash-doc`).

Comanda jobs

Comanda **jobs** afișează *job*-urile pentru terminalul curent:

```
razvan@asgard:~$ jobs
[1]   Running                 xemacs &
[2]   Running                 gnome-calculator &
[3]-  Stopped                 vi out.txt
[4]+  Stopped                 du -hs *

razvan@asgard:~$
```

Se poate observa că sunt patru *job*-uri dintre care două sunt în rulare (*Running*) iar celelalte două sunt suspendate (*Stopped*). Semnul + este folosit pentru *job*-ul curent (cel mai recent job). Semnul - este folosit pentru *job*-ul anterior *job*-ului curent.

Comenzile bg și fg

Utilizatorul poate determina rularea comenzilor care sunt suspendate (fie au fost pornite în fundal, fie au fost suspendate prin combinația de taste CTRL-Z). Aceste procese își pot continua execuția în *foreground* sau în *background*. Daca se dorește continuarea în *background* a execuției unui job suspendat, se folosește comanda **bg**; daca se dorește continuarea execuției în *foreground* a unui job suspendat, se folosește comanda **fg**.

În exemplul de mai jos, utilizatorul a decis continuarea execuției job-ului [4] în background. Rularea comenzii **bg** fără niciun parametru are ca efect continuarea job-ului curent. Daca se dorește rularea job-ului n, se folosește argumentul %n (spre exemplu **bg %5** pentru rularea în background a job-ului cu indicele 5).

```
razvan@asgard:~$ bg %4
[4]+ du -hs * &
1.2G    school
2.5M    shared-projects
1.7G    vmware

razvan@asgard:~$ jobs
[1]   Running                 xemacs &
[2]   Running                 gnome-calculator &
[3]+  Stopped                 vi out.txt
[4]-  Done                    du -hs *

razvan@asgard:~$ jobs
[1]   Running                 xemacs &
[2]-  Running                 gnome-calculator &
[3]+  Stopped
```

Se poate observa că s-a continuat rularea job-ului [4] în background după care procesul asociat s-a terminat.

Mesajul Done este afișat la încheierea rulării unor job-uri. Spre exemplu, daca vom închide din mediul grafic gnome-calculator și xemacs, mesajele afișate vor fi asemănătoare cu cele de mai jos:

```
razvan@asgard:~$
[1]   Done                    xemacs
[2]-  Done                    gnome-calculator
```

```
4
5 razvan@asgard:~$ jobs
6 [3]+  Stopped                 vi out.txt
```

După terminarea celor două job-uri, mai rămâne în background doar job-ul 3 (cel asociat editorului `vi`). Având în vedere că este singurul proces rămas, `vi` poate fi adus în foreground utilizând comanda **`fg`** sau **`fg %3`**:

```
1 razvan@asgard:~$ fg
2 vi out.txt
```

După rularea acestei comenzi, utilizatorul poate continua editarea fișierului `out.txt`.

5.3.4 Daemoni

> Un **daemon** este un tip particular de proces care rulează în fundal. Față de procesele care rulează în fundal (pornite prin intermediul operatorului &) sau care sunt suspendate în fundal (prin intermediul combinației `CTRL-Z`), un daemon se detașează de terminalul de control.

Astfel, un daemon nu va putea comunica direct cu utilizatorul prin intermediul terminalului: nu va accepta comenzi de control de la tastatură și nu va afișa rezultate la terminal. Comunicarea cu un daemon se face prin mecanisme mai complicate (ca de exemplu semnale – vezi secțiunea 5.4).

Inițial denumirea de daemon nu a avut nicio semnificație. Ulterior a fost găsită o abreviere pentru aceasta (*backronym*) de la *Disk And Execution MONitor*.

De obicei, procesele care sunt daemoni au numele terminat în d: `hald`, `udevd`, `sshd`, `inetd` etc. (fără a fi însă o regulă).

Pentru selectarea proceselor care nu au atașate niciun terminal (și ca atare pot fi daemoni), se folosește **`ps`** cu argumentul `-t` (pentru selecția după terminal) urmat de opțiunea - (semnul minus înseamnă procesele care nu au atașate niciun terminal).

```
1  razvan@asgard:~$ ps -t -
2    PID TTY          TIME CMD
3      1 ?        00:00:00 init
4      2 ?        00:00:00 ksoftirqd/0
5      3 ?        00:00:00 events/0
6  [..]
7   2657 ?        00:00:00 exim4
8   2665 ?        00:00:00 fbguard
9   2666 ?        00:00:00 fbserver
10  2676 ?        00:00:00 inetd
11  2689 ?        00:00:00 sshd
12  2695 ?        00:00:00 vsftpd
13 [...]
14  3006 ?        00:00:02 gedit
15  3009 ?        00:00:00 gnome-screensav
16  3016 ?        00:00:00 gnome-terminal
17  3017 ?        00:00:00 gnome-pty-helper
```

Se observă că nu toate procesele care nu au atașate un terminal sunt procese daemon. Spre exemplu, procesul gedit (pid 3006) nu este un daemon în adevăratul

sens al cuvântului. Deși nu are atașat niciun terminal de control, se poate comunica prin intermediul mediului grafic. În plus față de a nu avea asociat niciun terminal, un proces daemon va rula în background, eliminând posibilitatea comunicării directe (tastatură/mouse) cu utilizatorul.

Interacțiunea cu procesele daemon

Se pune însă întrebarea cum poate utilizatorul să interacționeze cu procesele daemon, în măsura în care nu o poate face prin intermediul intrării și ieșirii standard. Cele două mecanisme care permit acest lucru sunt fișierul/fișierele de configurare și semnalele.

În general, procesele daemon au unul sau mai multe fișiere de configurare. De exemplu:

- server-ul web **apache2** folosește `/etc/apache2/apache2.conf`;
- server-ul SSH **openssh** folosește `/etc/ssh/sshd_config`;
- server-ul **X** (mediul grafic) folosește `/etc/X11/xorg.conf`;

Aceste fișiere de configurare sunt citite în momentul creării procesului daemon și permit controlul funcționării acestuia. De exemplu, în cazul server-ului openssh, fișierul de configurare permite controlul tipului de autentificare (cu parolă sau prin chei publice), portul pe care serverul ascultă, dacă serverul permite X forwarding etc.

Dacă se dorește alterarea comportamentului server-ului, se editează fișierul de configurare. După salvarea fișierului, trebuie repornit procesul daemon, acesta va citi fișierul de configurare și va stabili noul comportament conform configurației din fișier.

Oprirea, suspendarea și repornirea unui proces, în general, și a unui daemon, în particular, se realizează cu ajutorul semnalelor. Daca în cazul unui proces care nu este daemon, acesta poate fi oprit sau suspendat și prin intermediul terminalului (prin combinații de taste), un proces daemon poate fi oprit/suspendat/repornit doar cu ajutorul semnalelor. Despre semnale se va discuta în secțiunea 5.4.

Comanda nohup

Comanda **nohup** poate fi utilizată pentru a rula un proces cu caracteristici de daemon. Utilizarea acestei comenzi determină ignorarea de către proces a semnalului SIGHUP (acest semnal este trimis de un terminal către procesele sale copil atunci când este închis). **nohup** este folosit, de obicei, atunci când utilizatorul va dori să părăsească terminalul dar comanda să fie rulată în continuare. Comanda **nohup** este urmată de comanda ce va fi rulată pentru pornirea procesului și de caracterul `&`. Un exemplu este prezentat în continuare:

```
1 asgard:/home/razvan# nohup updatedb &
2 [1] 3116
3
4 asgard:/home/razvan# nohup: appending output to 'nohup.out'
5
6 asgard:/home/razvan# ps -e
7 [...]
8  3116 pts/1    00:00:00 updatedb
9  3124 pts/1    00:00:00 updatedb
```

```
10  3125 pts/1    00:00:00 find
11  3126 pts/1    00:00:00 sort
12  3127 pts/1    00:00:00 frcode
13  3129 pts/1    00:00:00 ps
```

Se observă că procesul creat (`updatedb`) nu pierde controlul terminalului curent, însă nu se poate comunica cu acesta prin intermediul semnalelor de suspendare/repornire sau intrării/ieșirii standard. După cum reiese și din rezultatul rulării comenzii, ieșirea acesteia este redirecționată în fișierul `nohup.out`.

5.4 Semnale

Semnalele sunt mecanisme de notificare asincronă care sunt utilizate pentru a transmite o condiție specială unui proces.

Semnalele sunt asincrone în raport cu fluxul de rulare al procesului. Astfel, un semnal nu este corelat direct cu instrucțiunile executate de proces, putând fi transmis/primit la un moment de timp neștiut de proces.

Semnalele sunt transmise în două moduri:

- **de nucleu** (kernel) pentru a indica o condiție neobișnuită care solicită oprirea/suspendarea sau doar notificarea procesului;
- **de utilizator**.

Semnale care sunt transmise uzual de nucleu sunt următoarele (aceste semnale se pot trimite și de către utilizator, dar cel mai adesea tunt trimise de către nucleu):

- `SIGSEGV` (*signal segment violation*): în momentul accesării invalide a unei zone de memorie;
- `SIGBUS` în momentul unei erori pe magistrala sistemului;
- `SIGFPE` (*signal floating point error*): în momentul apariției unei erori de virgulă mobilă;
- `SIGTERM` (*signal termination*): pentru oprirea unui proces;
- `SIGKILL` (*signal kill*): pentru oprirea necondiționată a unui proces.

Semnale care sunt transmise uzual de utilizator sunt:

- `SIGINT` (*signal interrupt*): întrerupe procesul curent;
- `SIGQUIT` (*signal quit*): semnalează oprirea procesului curent;
- `SIGSTOP` (*signal stop*): suspenda procesul curent;
- `SIGCONT` (*signal continue*): repornește procesul suspendat.

Nu vom insista pe modul și condițiile în care nucleul transmite semnale unui proces, ci pe modul în care utilizatorul le transmite. Utilizatorul poate transmite unui proces semnale în două moduri:

- prin intermediul unor comenzi specifice: **`kill`**, **`killall`**; (vezi secțiunea 5.4.2)

- prin intermediul unor combinații de taste specifice. (vezi secțiunea 5.4.3)

Informații detaliate despre semnale se pot afla prin consultarea paginii de manual (**`man 7 signal`**).

5.4.1 Semnale importante UNIX

Înainte de a discuta despre modul de interacțiune a utilizatorului cu procesele prin intermediul semnalelor, trebuiesc amintite câteva semnale importante (unele menționate și anterior). Lista completă cu semnalele pe care le oferă sistemul de operare poate fi vizualizată folosind comanda **`kill`** cu opțiunea `-l` (*list*):

```
razvan@asgard:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Se observă că un semnal are un număr de identificare și un nume care oferă indicații despre efectul său asupra proceselor. Semnalele relevante sunt cele până la 31 (`SIGSYS`). Celelalte (`SIGRTMIN`, `SIGRTMAX`) sunt semnale de timp real (*SIGnal Real Time*) și nu sunt relevante pentru prezentarea curentă. Vom prezenta câteva dintre semnalele cele mai importante:

- `SIGHUP` (1) (*signal hang up*): este folosit pentru repornirea unui proces; acest semnal este de obicei transmis proceselor daemon pentru repornirea acestora și recitirea fișierului/fișierelor de configurare;
- `SIGINT` (2) (*signal interrupt*): este folosit pentru a întrerupe un proces; de obicei, efectul imediat este terminarea procesului;
- `SIGQUIT` (3) (*signal quit*): este folosit pentru închiderea unui proces; este mai puternic decât `SIGINT`: unele procese pot fi terminate cu `SIGQUIT` dar nu cu `SIGINT`;
- `SIGILL` (4) (*signal illegal*): procesul execută o instrucțiune invalidă;
- `SIGKILL` (9) (*signal kill*): cel mai puternic semnal; termină în mod necondiționat un proces
- `SIGSEGV` (11) (*signal segment violation*): semnal transmis în momentul accesului invalid la o zona de memorie; rezultă, de obicei, în terminarea procesului și la apariția mesajului “Segmentation fault”;

- `SIGTERM` (15) (*signal termination*): termină un proces; în mod tipic nucleul transmite întâi `SIGTERM` unui proces pentru a-l anunța că va fi terminat; procesul realizează operații de curățare (*cleanup*), după care kernel-ul transmite `SIGKILL` și procesul este terminat;
- `SIGSTOP` (19) (*signal stop*): suspendă procesul curent;
- `SIGCONT` (18) (*signal continue*): reia execuția unui proces suspendat.

5.4.2 Comenzile kill, killall și pkill

Comenzile **`kill`**, **`killall`** și **`pkill`** sunt utilizate pentru a transmite un semnal unui proces sau unui set de procese.

Comanda kill

Comanda **`kill`** transmite un semnal unui proces. Comanda primește ca argument pid-ul procesului (sau pid-urile proceselor) către care se dorește să se transmită semnalul. În mod implicit comanda transmite semnalul `SIGTERM`:

```
1  razvan@asgard:~$ xcalc &
2  [1] 3590
3
4  razvan@asgard:~$ ps
5    PID TTY          TIME CMD
6   3092 pts/1    00:00:00 bash
7   3590 pts/1    00:00:00 xcalc
8   3591 pts/1    00:00:00 ps
9
10 razvan@asgard:~$ kill 3590
11
12 razvan@asgard:~$
13 [1]+  Terminated              xcalc
14
15 razvan@asgard:~$ ps
16   PID TTY          TIME CMD
17  3092 pts/1    00:00:00 bash
18  3592 pts/1    00:00:00 ps
```

Comanda poate fi configurată pentru a transmite și alt semnal în afară de `SIGTERM`. Astfel, dacă se dorește transmiterea semnalului `SIGSTOP` către procesele cu pid-urile 1234 și 5678, se poate utiliza una din alternativele:

```
1  razvan@asgard:~$ kill -SIGSTOP 1234 5678
2
3  razvan@asgard:~$ kill -STOP 1234 5678
4
5  razvan@asgard:~$ kill -19 1234 5678
```

Se observă că semnalul poate fi transmis ca parametru fie în formă literală (`SIGSTOP`, `STOP`), fie în formă numerică (19).

Comanda killall

Comanda **killall** este folosită pentru transmiterea unui semnal către un proces sau către un set de procese atunci când se cunoaște numele procesului. Un exemplu de utilizare este prezentat mai jos:

```
root@asgard:/home/razvan# ps -C apache2 -H
  PID TTY          TIME CMD
 2792 ?        00:00:00 apache2
 2793 ?        00:00:00   apache2
 2794 ?        00:00:00   apache2
 2796 ?        00:00:00   apache2

root@asgard:/home/razvan# killall -KILL apache2

root@asgard:/home/razvan# ps -C apache2
  PID TTY          TIME CMD
```

În acest exemplu, s-a trimis semnalul de terminare a unui proces (`SIGKILL`) către toate procesele apache2 din sistem. Modul de transmitere a semnalului ca argument al comenzii este același ca în cazul comenzii **kill**.

O opțiune care poate fi utilă este opțiunea `--user` care permite transmiterea unui semnal către toate procesele unui anumit utilizator. Pentru a putea utiliza această opțiune, utilizatorul care apelează **killall** trebuie să fie `root`:

```
root@asgard:/home/razvan# ps -o pid,comm,user -u guest
  PID COMMAND         USER
 3761 su              guest
 3762 bash            guest
 3802 su              guest
 3803 bash            guest
 3822 top             guest
 3825 bc              guest

asgard:/home/razvan# killall --user guest

asgard:/home/razvan# ps -o pid,comm,user -u guest
  PID COMMAND         USER
```

În situația de mai sus s-a transmis semnalul implicit (`SIGTERM`) către toate procesele utilizatorului guest, rezultând în terminarea acelor procese.

Comanda pkill

Comanda **pkill** poate fi comparată cu comanda **pgrep**. În timp ce comanda **pgrep** oferea informații despre procesele care îndeplineau anumite criterii, comanda **pkill** transmite un semnal proceselor pe baza unor criterii de selecție a acestora.

La fel ca în cazul **pgrep**, selecția se poate realiza după pid-ul procesului părinte, după terminalul utilizat, după id-ul utilizatorului sau al grupului. Argumentul semnal se transmite la fel ca în cazul comenzilor **kill** și **killall**.

5.4.3 Transmiterea de semnale prin combinații de taste

În afara comenzilor de mai sus, un utilizator poate transmite anumite semnale procesului curent (cel care rulează în acel moment în terminal) prin intermediul unor combinații de taste specifice.

Astfel de combinații sunt:

- `CTRL-Z`: transmite semnalul `SIGSTOP` către procesul curent, care are drept consecință suspendarea acestuia. O situație utilă este atunci când se rulează editorul *vi* și se dorește rularea altei comenzi. Se folosește `CTRL-Z` pentru suspendarea procesului vi și revenirea la promptul terminalului, unde poate fi introdusă noua comandă. După introducerea comenzii se folosește comanda **`fg`** pentru a reactiva procesul vi.
- `CTRL-C`: transmite semnalul `SIGINT` către procesul curent, care, în general, va omorî procesul. De obicei, combinația de taste se folosește în cazul în care un proces este blocat.
- `CTRL-\`: transmite semnalul `SIGQUIT`. Acesta are același rol ca și `CTRL-C`, doar că este mai puternic.

Trebuie precizat că nu toate aceste combinații de taste au efect pentru toate procesele, întrucât unele procese pot **ignora semnalele** transmise (pot avea un comportament non-standard la primirea unui semnal).

5.5 Comunicația între procese

La fel cum în cadrul unui departament mai mulți oameni conlucrează pentru a îndeplini o sarcină, și în cadrul unui sistem de operare procesele acestuia interacționează. Interacțiunea proceselor într-un sistem de operare poartă numele de **comunicație între procese**.

De ce este nevoie de comunicație între procese? În primul rând pentru ca un proces poate avea nevoie de resursele pe care i le pune la dispoziție un alt proces. Spre exemplu, în cazul utilitarului **`top`**, acesta culege informații pe care i le furnizează alte procese prin intermediul nucleului sistemului de operare și al sistemului de fișiere procfs.

În altă situație, un proces A trebuie să aștepte ca un alt proces B să termine o sarcină pentru ca A să poată continua. Aceasta formă de comunicație între A și B se numește sincronizare: un proces trebuie să aștepte un alt proces.

Comunicația între procese este intermediată în diverse moduri: cu ajutorul unui fișier, a unui socket, a unei zone de memorie sau a unui canal de comunicație (pipe). Intermedierea printr-un fișier este destul de simplă: un proces scrie o informație într-un fișier, iar un alt proces o citește. Nu vom prezenta comunicația prin intermediul unui socket sau a unei regiuni de memorie, întrucât depășesc aria de cuprindere a acestui capitol. Vom discuta, însă, despre comunicația prin intermediul unui canal de comunicație (pipe).

5.5.1 Operatorul | (pipe)

Operatorul | (**pipe**) este folosit pentru a asigura comunicația între două procese rulate din linia de comandă, folosind un canal de comunicație numit *pipe*.

Comunicația prin intermediul unui pipe este o îmbunătățire a comunicației prin intermediul unui fișier. Astfel, în cazul comunicației prin intermediul unui fișier, un proces scrie datele de ieșire într-un fișier iar un alt proces folosește acel fișier ca intrare. În cazul comunicației prin pipe, ieșirea primului proces este folosită direct ca intrare pentru al doilea, fără a mai fi nevoie de un fișier pe disc.

Vom folosi pentru exemplificare comanda **`grep`**, care caută un cuvânt în cadrul unui fișier. Dacă, spre exemplu, dorim aflarea de informații despre procesele bash din sistem folosind **`grep`**, ar trebui, în primă fază, să redirectăm ieșirea comenzii **`ps`** într-un fișier, iar apoi să folosim **`grep`** pe acel fișier, ca în exemplul de mai jos:

```
1  razvan@asgard:~$ ps -e > ps.out
2
3  razvan@asgard:~$ grep bash ps.out
4   3029 pts/0    00:00:00 bash
5   3092 pts/1    00:00:00 bash
6   3299 pts/2    00:00:00 bash
7   3317 pts/2    00:00:00 bash
8   3740 pts/3    00:00:00 bash
9   3758 pts/3    00:00:00 bash
10  3781 pts/4    00:00:00 bash
11  3799 pts/4    00:00:00 bash
```

Același rezultat ca mai sus poate fi însă realizat prin intermediul operatorului |, fără a mai fi nevoie de intermedierea printr-un fișier:

```
1  razvan@asgard:~$ ps -e | grep bash
2   3029 pts/0    00:00:00 bash
3   3092 pts/1    00:00:00 bash
4   3299 pts/2    00:00:00 bash
5   3317 pts/2    00:00:00 bash
6   3740 pts/3    00:00:00 bash
7   3758 pts/3    00:00:00 bash
8   3781 pts/4    00:00:00 bash
9   3799 pts/4    00:00:00 bash
```

Pentru a exemplifica puterea acestui operator vom considera următoarea problemă: dorim să aflăm primii 3 utilizatori din sistem care au directorul de bază în `/home`, ordonați în ordinea alfabetică a numelui de utilizator.

Informații despre utilizatorii din sistem se găsesc în `/etc/passwd`. Vom folosi grep pentru a extrage utilizatorii care au directorul de bază în `/home`:

```
1  razvan@asgard:~$ cat /etc/passwd | grep /home
2  razvan:x:1000:1000:razvan,,,:/home/razvan:/bin/bash
3  haldaemon:x:105:106:Hardware abstraction layer,,,:/home/haldaemon:/bin/
   false
4  guest:x:1001:1001:Guest Account,,,:/home/guest:/bin/bash
5  ftp:x:107:65534::/home/ftp:/bin/false
```

În continuare vom sorta rezultatul obținut, după care vom reține numai primele 3 intrări:

```
1  razvan@asgard:~$ cat /etc/passwd | grep /home | sort
```

```
ftp:x:107:65534::/home/ftp:/bin/false
guest:x:1001:1001:Guest Account,,,:/home/guest:/bin/bash
haldaemon:x:105:106:Hardware abstraction layer,,,:/home/haldaemon:/bin/
false
razvan:x:1000:1000:razvan,,,:/home/razvan:/bin/bash

razvan@asgard:~$ cat /etc/passwd | grep /home | sort | head -3
ftp:x:107:65534::/home/ftp:/bin/false
guest:x:1001:1001:Guest Account,,,:/home/guest:/bin/bash
haldaemon:x:105:106:Hardware abstraction layer,,,:/home/haldaemon:/bin/
false
```

5.6 Swapping

Un concept important în studiul sistemelor de operare și al proceselor acestora este conceptul de **swapping**. Swapping este o măsură compensare a cantității insuficiente de memorie RAM de care dispune un sistem la un moment dat. Pentru compensare, sistemul de operare va folosi o porțiune din disc pentru stocarea datelor din RAM.

În cazul unui sistem încărcat (cu multe procese), memoria RAM se poate dovedi insuficientă pentru a satisface toate procesele. În această situație o parte din paginile de memorie din RAM sunt evacuate (*swapped*) pe disc pentru a face loc pentru paginile de memorie utile în momentul de față. În momentul în care paginile evacuate sunt din nou necesare, se vor evacua alte pagini pentru a se aduce la loc cele curente.

Principiul swapping-ului este prezentat în figura de mai jos:

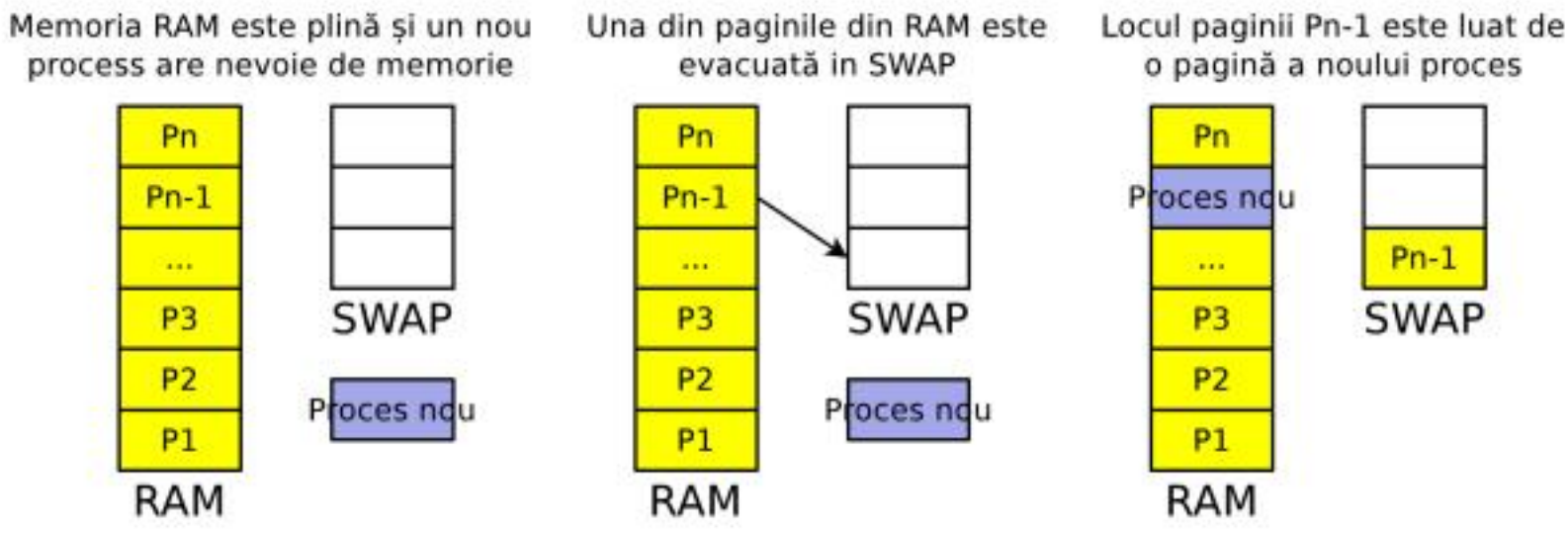


Figura 5.3: Swaping

Pe sistemele Linux swapping-ul este realizat în general pe partiții dedicate (partiția de swap – vezi secțiunea 2.2.2), care sunt folosite în cazul în care memoria din sistem este insuficientă. În cazul Windows, spațiul de swap alocat din partițiile sistemului, fiind utilizate în acest scop fișiere. Pentru a afla informații despre spațiul de swap utilizat în Linux se poate folosi comanda **`free`** (aceste informații se găsesc și în zona de sumarizare din ecranul `top`):

```
razvan@asgard:~$ free
             total       used       free     shared    buffers     cached
Mem:        514748     336876     177872          0      19892     185200
-/+ buffers/cache:     131784     382964
Swap:       497972          0     497972
```

Se observă că sistemul beneficiază de 512 MB de memorie RAM și nu este folosit spațiul de swap.

5.7 Studii de caz

5.7.1 Managementul proceselor/serviciilor pe Windows

În Windows, un proces are caracteristici asemănătoare cu procesele din Linux. Totuși, deși un proces este creat prin intermediul altui proces, nu există o ierarhie de procese și nici noțiunea directă de proces părinte și proces fiu. La fel ca în Linux, un proces este identificat printr-un PID, are asociate zone de memorie și fișiere deschise etc.

Task manager

Interfața de vizualizare și gestiune a proceselor în Windows este asigurată de Task Manager. Pentru pornirea acestuia se folosește combinația de taste `CTRL-ALT-DEL` sau `CTRL-SHIFT-ESC` sau click dreapta pe bara de task-uri și selectarea opțiunii `Task Manager` din meniu.

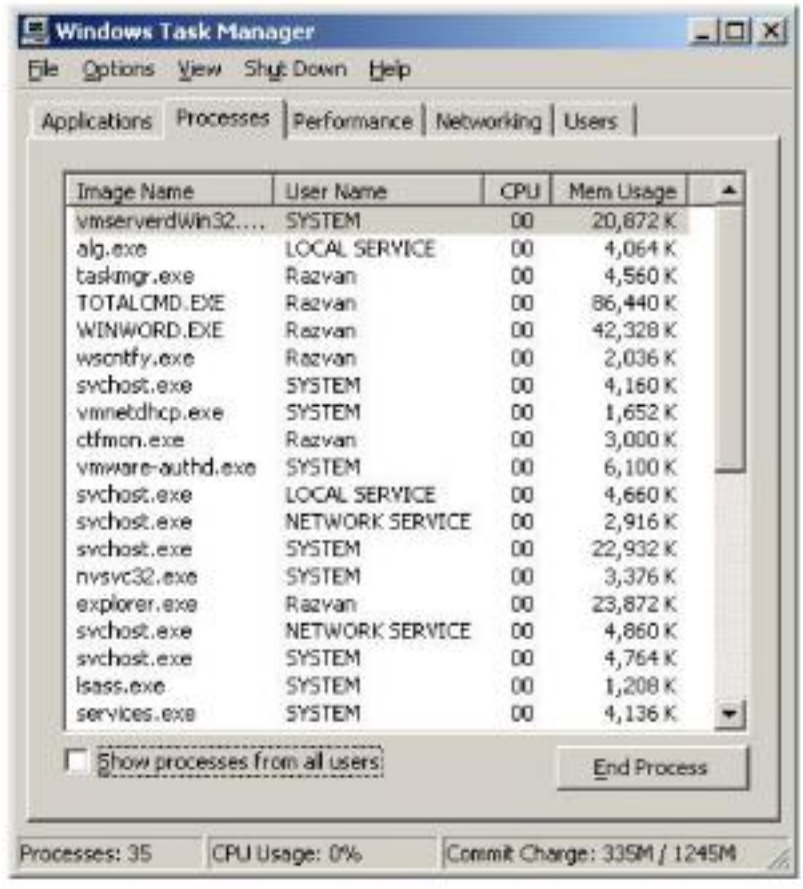


Figura 5.4: Windows Task Manager

Task Manager poate fi comparat cu utilitarul top. Afișează informații despre procese și despre sistem și poate fi configurat pentru personalizarea afișării sau pentru interacțiunea cu procesele din sistem.

În mod implicit, Task Manager afișează numele programului care a generat procesul, utilizatorul care a creat procesul, procentajul de procesor folosit și memoria utilizată. Dacă se dorește afișarea altor câmpuri se poate accesa meniul `View->Select`

`Columns`. Se pot afișa PID-ul procesului, memoria virtuală utilizată, maximul de memorie utilizat, numărul de fișiere deschise, numărul de thread-uri etc.

În afara vizualizării proceselor din sistem, Task Manager mai oferă următoarele servicii:

- permite crearea unui nou proces prin introducerea imaginii de executabil asociate (`File->New Task (Run...)`);
- permite monitorizarea și analiza performanței sistemului, prin accesarea tab-ului `Performance`; informațiile de aici sunt asemănătoare cu cele oferite de top în zona de sumarizare;
- permite deconectarea unui utilizator, sau oprirea, repornirea sau suspendarea sistemului (meniul `Shut Down`);
- permite terminarea unui proces; se selectează procesul dorit și se apasă butonul `End Process`.

Se poate observa că Windows Task Manager oferă cea mai mare parte din facilitățile furnizate de comenzile Unix de interacțiune cu procese sistemului. Un utilitar puternic de vizualizare și control a proceselor pe Windows este ProcessExplorer[1].

Services

În Windows, procesele daemon poartă numele de **servicii**. Ca și în Linux, serviciile sunt procese care rulează în *background* și sunt folosite pentru monitorizarea și întreținerea sistemului.

Fereastra de interacțiune cu serviciile Windows este afișată, în Windows XP, prin accesarea `Start->Control Panel->Administrative Tools->Services` sau click dreapta pe `My Computer->Manage->Services`.

Ecranul afișat prezintă toate serviciile sistemului. Coloanale afișate indică numele serviciului, o scurtă descriere, starea lui (oprit sau pornit), modul de pornire a serviciului (dezactivat, manual sau automat și, la Windows Vista, întarziat). Pentru pornirea, oprirea sau repornirea serviciului, ca și pentru schimbarea modului de pornire, se accesează meniul contextual al serviciului (click dreapta) și se selectează opțiunea `Properties`:

5.7.2 Procese importante

Orice sistem de operare deține un set de procese cu rol important în rularea și întreținerea sistemului de operare.

Procese importante Linux

Procese importante în Linux sunt:

[1] http://www.microsoft.com/technet/sysinternals/utilities/ProcessExplorer.mspx

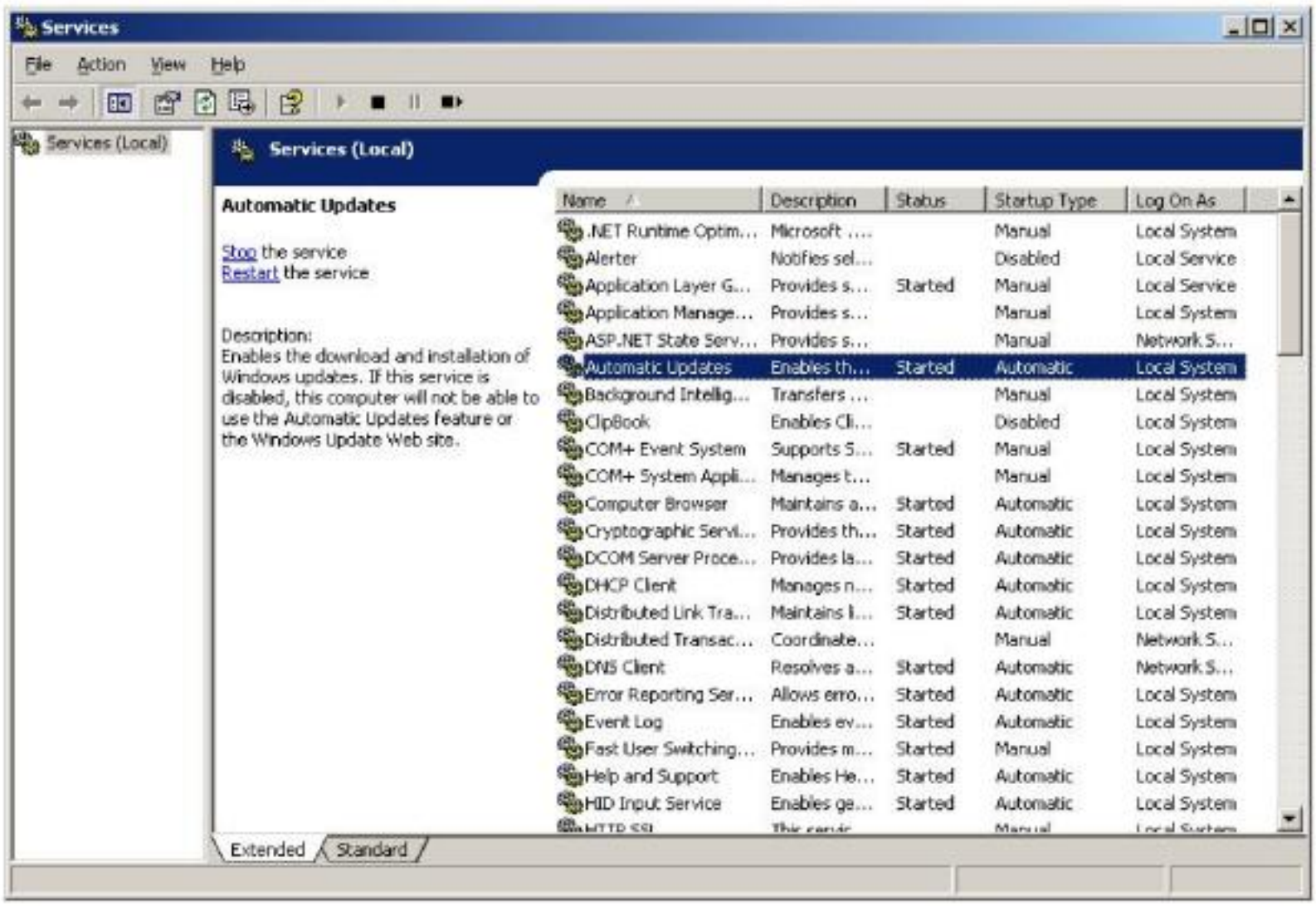


Figura 5.5: Windows Services

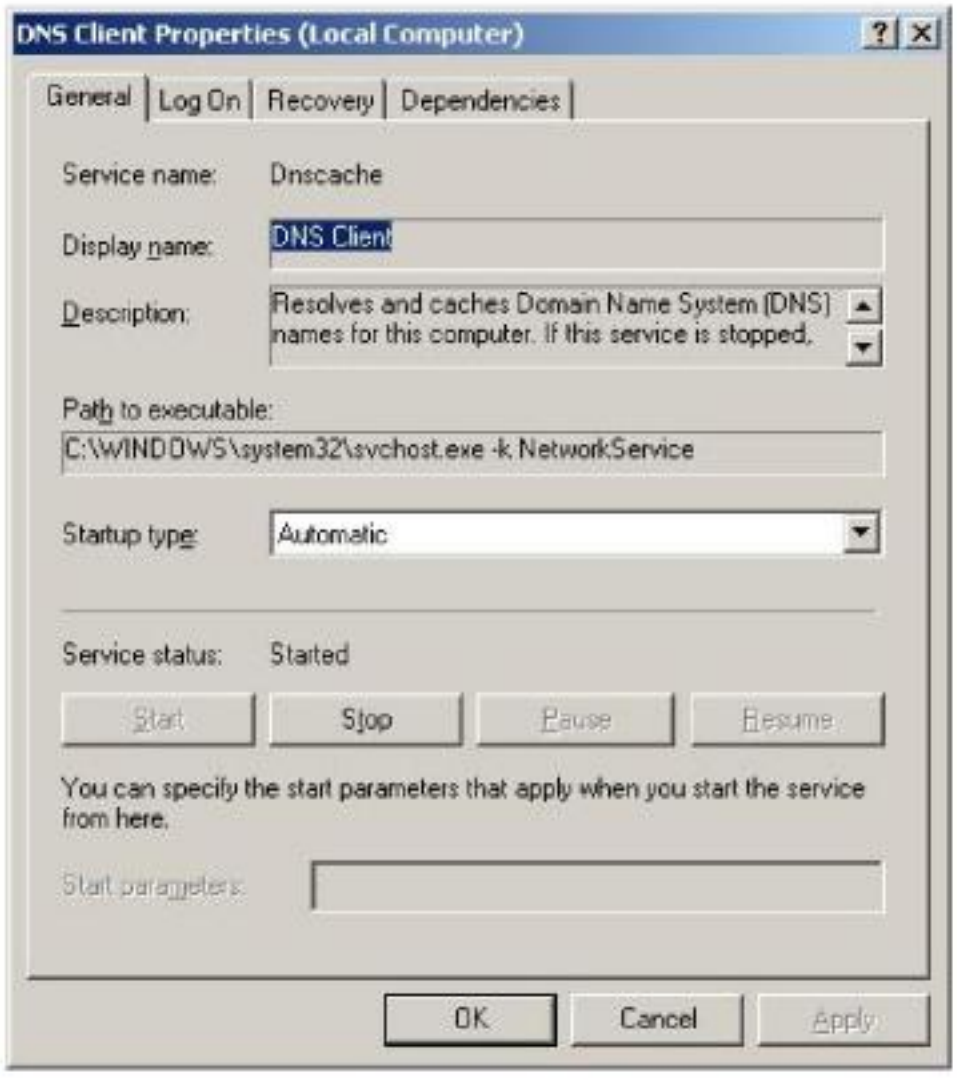


Figura 5.6: Interfața de control a serviciului Windows de DNS

- `init`: este procesul cu pid-ul 1 în sistemele Unix și este părintele tuturor proceselor; rolul său nu se limitează doar la a porni celelalte procese importante

în sistem ci și de a deveni procesul părinte pentru procesele rămase orfane;

- `kswapd` (kernel swap daemon): este responsabil cu alegerea paginilor de memorie care vor fi evacuate pe disc în momentul în care memoria este insuficientă;
- `pdflush` (*page daemon*): este responsabil cu asigurarea consistenței buffer-elor folosite de sitemul de fișiere;
- `getty`: este procesul resposabil cu autentificarea în sistem în terminalele virtuale; getty oferă promptul login : utilizatorului;
- `Xorg`: este procesul resposabil cu pornirea interfeței grafice și are asociat terminalul virtual tty7 (ce poate fi accesat prin combinația `ALT+CTRL-F7`).

Procese importante Windows

În Windows procese importante sunt:

- `lsass` (*Local Security and Authentication Server*): verifică validitatea autentificării utilizatorilor; în cazul unei autentificări corecte generează un jeton folosit pentru deschiderea unei prime sesiuni shell pe sistem; lsass este și numele unui virus;
- `svchost` (*Generic Host Process for Win32 Services*): este procesul folosit pentru gestiunea serviciilor din Windows;
- `csrss` (*Client/Server Run-time Subsystem*): este responsabil pentru crearea ferestrelor de consolă și gestiunea thread-urilor;
- `winlogon`: este o componentă esențială a subsistemului de autentificare în Windows;
- `explorer`: este este shell-ul unui sistem Windows, însă cu interfață grafică.

Mai multe informații despre procesele importante Windows găsiți la[1].

5.7.3 Prioritatea unui proces

În sistemele de operare moderne, procesele au asociată o **prioritate**. Aceasta influențează probababilitatea ca un proces să fie planificat pe procesor: cu cât un proces are o prioritate mai mare, cu atât probabilitatea ca el să fie următorul proces planificat pentru execuție crește. De obicei procesele care se blochează des (denumite I/O intensive) au asociate o prioritate mai mare, în timp ce procesele care utilizează intens procesorul (CPU intensive), au o prioritate mai mică.

Prioritatea proceselor nu este fixă. Sistemul de operare poate modifica prioritatea proceselor pentru a asigura echitatea planificării. Totodată, și utilizatorul poate modifica prioritatea unui proces dacă dorește acest lucru.

[1] http://www.neuber.com/taskmanager/process/

Schimbarea priorității unui proces în Linux

În Linux, prioritatea unui proces poartă numele de **niceness** (cât de echitabil se comportă un proces raportat la celelalte). Un proces are o prioritate mai mare cu cât valoarea sa nice este mai mică. Valorea nice pentru un proces se încadrează în limitele [-20, 19]. -20 înseamnă un proces maxim prioritar, în timp ce 19 înseamnă un proces minim prioritar.

Prioritatea unui proces într-un sistem Linux se află folosind **ps** și opțiunea `nice`:

```
razvan@asgard:~$ ps -e -o pid,tty,comm,nice --sort nice
  PID TT       COMMAND          NI
    3 ?        events/0         -5
    4 ?        khelper          -5
    5 ?        kthread          -5
    8 ?        kblockd/0        -5
    9 ?        kacpid           -5
  [...]
    1 ?        init              0
  [...]
 3799 pts/4    bash              0
 4009 pts/1    ps                0
    2 ?        ksoftirqd/0      19
```

Majoritatea proceselor pornesc cu prioritatea 0. Pentru a modifica prioritatea de start a unui proces se folosește comanda **nice**, folosind argumentul `-n`:

```
razvan@asgard:~$ nice -n -10 ps -e -o pid,tty,comm,ni
nice: cannot set niceness: Permission denied

razvan@asgard:~$ nice -n 15 ps -C ps -o pid,tty,comm,ni
  PID TT       COMMAND          NI
 4051 pts/1    ps               15

asgard:/home/razvan# nice -n -10 ps -C ps -o pid,tty,comm,ni
  PID TT       COMMAND          NI
 4048 pts/2    ps              -10
```

Se poate observa că un utilizator neprivilegiat nu poate stabili o prioritate negativă (puternică), dar poate stabili una pozitivă (mai slabă). Utilizatorul `root` poate modifica prioritarea unui proces în orice direcție.

Schimbarea în timp real a priorității unui proces se poate realizeaza prin intermediul **top**. În ecranul **top** se folosește tasta `r` (*renice*). Zona de comandă oferă promptul `PID to renice:` în care se introduce pid-ul procesului a cărui prioritate va fi schimbată. După aceasta se va cere introducerea unei valori nice asociată procesului. Un utilizator neprivilegiat poate doar incrementa valoarea nice a unui proces (îl va face mai puțin prioritar).

Schimbarea priorității unui proces în Windows

În Windows, utilizatorul poate stabili prioritatea unui proces la niște valori cu granularitate mai mare: High, Very High, Normal etc. Pentru aceasta se folosește tot interfața pusă la dispoziție de `Task Manager`. Pentru alterarea prioritații unui proces, se realizează

click dreapta pe intrarea asociată procesului, se alege opțiunea `Set priority` și apoi se stabilește prioritatea dorită. Un exemplu se poate găsi în figura de mai jos:

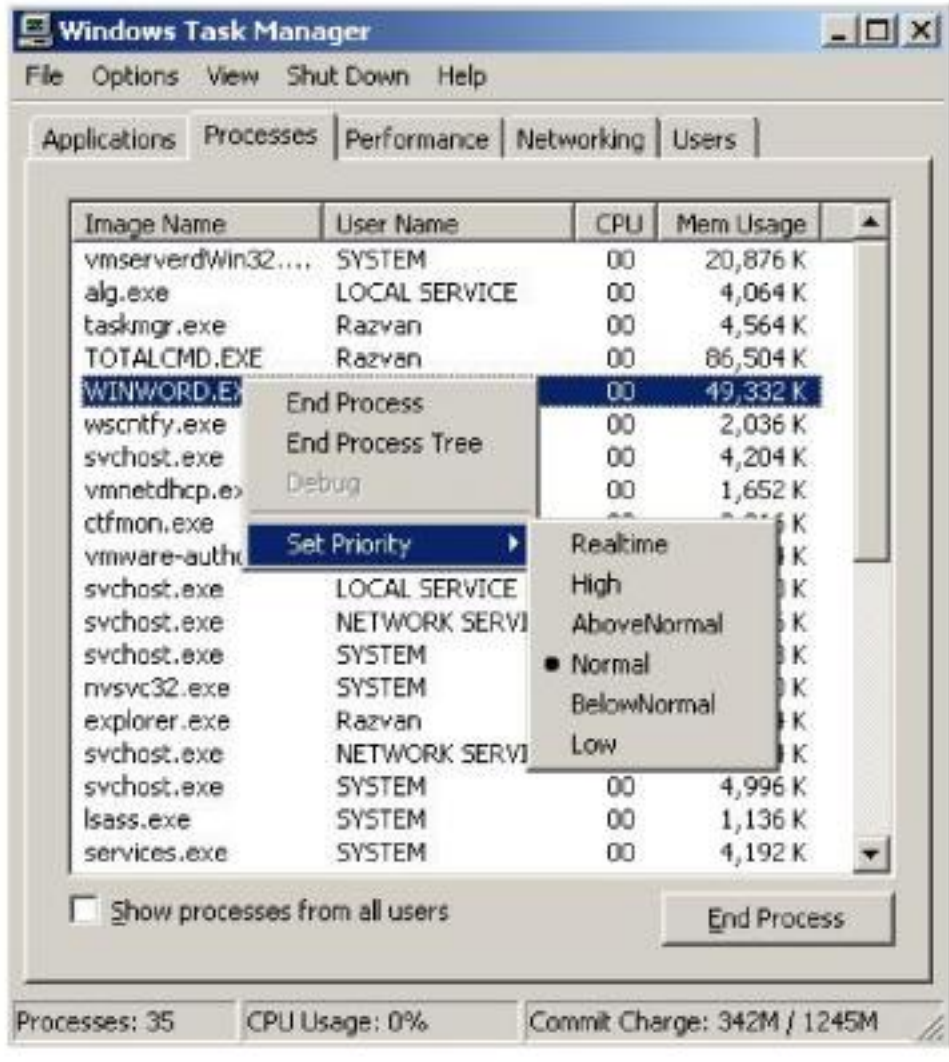


Figura 5.7: Stabilirea priorității unui proces în Windows

Cuvinte cheie

- proces
- program
- pid
- procfs
- interactivitatea unui proces
- context switch
- cuantă de timp
- multitasking
- ierarhie de procese
- init
- ps
- pstree
- pgrep
- top
- background
- foreground
- operatorul &
- bg, fg
- suspendarea unui proces
- terminal
- proces daemon
- nohup
- semnal
- kill
- killall, pkill
- CTRL-Z, CTRL-C, CTRL- CTRL-X

- operatorul |
- Task manager
- Services
- init, kswapd, pdflush, getty
- lsass, svchost, csrss
- prioritatea unui proces
- nice

Întrebări

1. Care din următoarele este un proces important Unix?
   - ❑ **ps**
   - ❑ SIGQUIT
   - ❑ **init**
   - ❑ **lsass**
2. Care utilitar NU poate fi folosit pentru a transmite semnale către un proces?
   - ❑ **kill**
   - ❑ **top**
   - ❑ **nohup**
   - ❑ **killall**
3. Câte procese **init** pot exista la un moment dat într-un sistem Linux?
   - ❑ 1
   - ❑ câte unul per procesor
   - ❑ niciunul
   - ❑ oricâte
4. Câte procese bash pot exista la un moment dat într-un sistem Linux?
   - ❑ 1
   - ❑ câte unul per procesor
   - ❑ niciunul
   - ❑ oricâte
5. Care din următoarele comenzi NU afișează PID-ul unui proces?
   - ❑ **ps**
   - ❑ **pgrep**
   - ❑ **kill**
   - ❑ **top**
6. În Linux orice program poate fi imaginea unui singur proces. Comanda top NU afișează PID-urilor proceselor din sistem.

- ❑ adevărat, adevărat
- ❑ adevărat, fals
- ❑ fals, adevărat
- ❑ fals, fals

7. Care comandă poate produce aceleași efecte ca apăsarea combinației de taste `CTRL-Z`?
   - ❑ **`pkill`**
   - ❑ **`ps`**
   - ❑ operatorul `&`
   - ❑ **`fg`**
8. Ce director din sistemul de fișiere oferă informații despre procese?
   - ❑ `/sys`
   - ❑ `/proc`
   - ❑ `/home`
   - ❑ `/`
9. În Linux procesele NU au pid-uri negative. În Linux procesele NU au priorități negative.
   - ❑ adevărat, adevărat
   - ❑ adevărat, fals
   - ❑ fals, adevărat
   - ❑ fals, fals
10. Care asociere NU este validă?
    - ❑ Task Manager – top
    - ❑ serviciu – daemon
    - ❑ explorer – nice
    - ❑ lsass – getty

Capitolul 6

Pornirea și inițializarea sistemului

Press Ctrl-Alt-Del now for IQ test.

Ce se învață din acest capitol?

- Pornirea sistemului; conceptul de bootstrapping
- Noțiunea de multitasking
- Fazele pornirii sistemului
- BIOS (POST, CMOS)
- Dispozitive boot-abile, sector bootabil (MBR/VBR)
- Rolul unui bootloader
- GRUB: rulare și configurare
- Încărcarea nucleului Linux
- Inițializarea unui sistem Linux: init/upstart și pornirea serviciilor
- Pornirea și inițializarea Windows
- Noțiuni de interoperabilitate Linux-Windows

6.1 Pornirea sistemului

Pornirea și inițializarea sistemului se referă la mecanismele prin care sistemul de operare și aplicațiile de bază ajung să fie încărcate și folosite de utilizator.

Pornirea sistemului se realizează, de obicei, prin apăsarea unui buton specific de pe unitatea centrală. În sens mai larg, ne vom referi la pornirea sistemului ca fiind o acțiune compusă ce cuprinde atât inițializarea hardware cât și cea software.

Inițializarea hardware presupune verificarea și configurarea componentelor hardware. Aceasta se execută în două situații: atunci când sistemul trece din starea oprit în starea pornit (*power on*) și atunci când sistemul este reinițializat (*reset*, *reboot*).

Acțiunea de *power on* se referă la activarea sistemului din momentul în care acesta era oprit, de cele mai multe ori prin intermediul butonului de pornire[1]. Repornirea sistemului se referă la reactivarea sistemului: din starea pornit, eventual rulând o instanță a unui sistem de operare, sistemul este repornit (fie prin intermediul butonului *Reset*, fie printr-o comandă dată sistemului de operare).

Inițializarea software presupune încărcarea sistemului de operare. Nu vom considera rularea anumitor aplicații peste sistemul de operare ca făcând parte din inițializarea software.

Pornirea sistemului prezintă un set de etape care vor fi precizate în continuare.

6.1.1 Problematica pornirii sistemului – bootstrapping

> Pornirea sistemului fizic și încărcarea sistemului de operare poartă numele de **booting** sau de **bootstrapping**.

Denumirea de *bootstrapping* își are originea în Baronul de Münchhausen. Într-o poveste a acestuia, el reușește să se salveze de la înec trăgându-se de baierele cizmelor (*boot* = cizmă, *strap* = baieră). Analogia cu această poveste se reflectă în faptul că sistemul de operare trebuie încărcat fără existența unui alt sistem de operare.

Un procesor poate executa numai cod aflat în memoria ROM[2] sau în memoria RAM. Atunci când o aplicație trebuie executată, sistemul de operare muta codul acesteia în memoria RAM și procesorul începe execuția aplicației.

Apare astfel întrebarea: cum se încarcă în memorie sistemul de operare, dacă nu există un alt sistem de operare care să îl încarce ? Aparent, trebuie "să ne tragem de baierele cizmelor" pentru "a ne scoate din apă". Vom vedea, însă, că există un set de etape care se vor parcurge succesiv pentru încărcarea diverselor componente până la încărcarea completă a sistemului de operare.

6.1.2 Etapele pornirii sistemului

Pornirea sistemului presupune încărcarea în memorie a diverselor componente până la încărcarea completă a sistemului de operare. Așa cum se poate vedea și în figura 6.1, etapele importante ale pornirii sistemului sunt:

- **încărcarea BIOS-ului**: presupune încărcarea unui program de mici dimensiuni, aflat într-o memorie dedicată de pe placa de bază, și care este utilizat pentru inițializarea componentelor hardware;

[1] Alternativ față de apăsarea butonului de pornire, un sistem poate fi pornit și de la distanță folosind WOL – http://en.wikipedia.org/wiki/Wake-on-LAN

[2] http://en.wikipedia.org/wiki/Read-only_memory

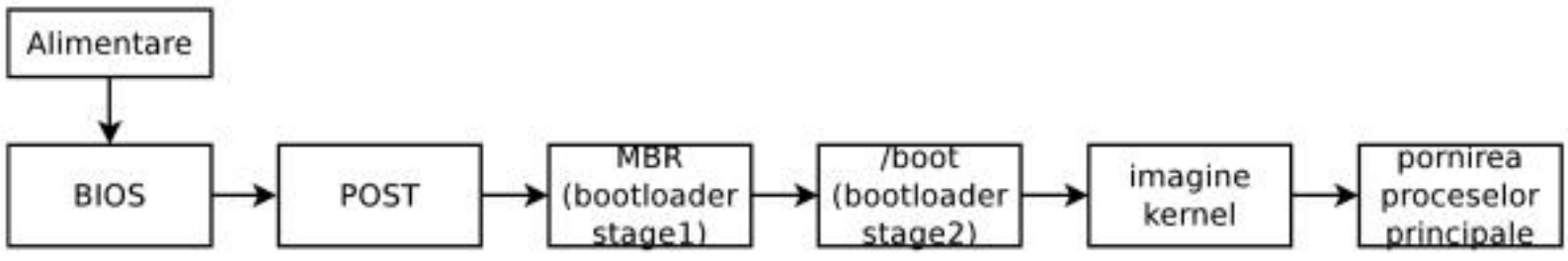


Figura 6.1: Etapele pornirii sistemului

- **rularea POST**: presupune realizarea unui set de teste hardware ce verifică funcționalitatea de bază a componentelor principale;
- **încărcarea bootloader-ului**: presupune încărcarea unui program de mici dimensiuni destinat copierii codului kernelului într-o zonă din RAM de unde să poată fi executat de procesor;
- **încărcarea nuclelui**: presupune un set de inițializări de bază pe care le realizează nucleul (kernel-ul) sistemului de operare;
- **inițializarea sistemului de operare**: presupune pornirea principalelor procese care asigura utilizarea sistemului de operare.

Etapele prezentate sunt ordonate "cronologic": o etapă este rulată și, la finalul ei, este responsabilă pentru rularea etapei ulterioare. Vor fi descrise în cadrul acestui capitol fiecare din fazele prezentate și interacțiunea între acestea.

6.1.3 BIOS

BIOS-ul (*Basic Input Output System*) este primul program încărcat în momentul pornirii sistemului. BIOS-ul este responsabil cu verificarea și inițializarea componentelor hardware.

BIOS-ul este un exemplu de **firmware**. Firmware-ul este o componentă software distribuită sub formă binară pe un suport de memorie nevolatilă. În cazul de față, BIOS-ul este stocat pe un chip de memorie ROM de pe placa de bază[1]. La *power on* sau la *reboot* BIOS-ul este rulat automat din chip-ul de pe placa de bază. BIOS-ul cunoaște specificul hardware al sistemului și devine responsabil cu inițializarea componentelor fizice ale acestuia.

CMOS

Deși este un program mic și limitat la specificul hardware al sistemului, BIOS-ul oferă optiuni de configurare. Configurările BIOS-ului sunt păstrate într-un chip separat de memorie RAM nevolatilă alimentată prin intermediul unei baterii. Aceast chip de memorie se numește CMOS[2] – *Complementary Metal-Oxide Semiconductor*. Atât chip-ul cât și bateria se găsesc pe placa de bază. În momentul în care bateria este scoasă, configurările BIOS-ului se pierd revenindu-se la configurările implicite.

[1] Memoria este de tip EPROM – http://en.wikipedia.org/wiki/EPROM
[2] http://en.wikipedia.org/wiki/Cmos

Figura 6.2: Ecran de configurare CMOS

Configurarea BIOS-ului se realizează prin intermediul unui ecran de configurare. Opțiunile disponibile includ stabilirea unei parole pe BIOS, selectarea dispozitivului de boot (vezi secțiunea 6.2.1), alterarea frecvenței procesorului și alte configurări care depind de placa de bază și de componentele hardware existente în sistem.

6.1.4 POST

Etapa de verificare și de inițializarea componentelor hardware ale sistemului (procesor, memorie, placă video) poartă numele de POST (*Power-on Self Test*). Dacă una dintre componente prezintă erori și nu poate fi inițializată corespunzător, atunci BIOS-ul va emite un set de sunete specifice și nu se va continua inițializarea sistemului.

După verificarea și inițializarea componentelor, BIOS-ul este responsabil cu încărcarea bootloader-ului.

6.2 Bootloader

Bootloader-ul este componenta utilizată în principal pentru încărcarea imaginii de kernel într-o zonă din memori RAM, de unde să poată fi executată de către procesor. Cele mai multe *bootloader*-e permit utilizatorului să aleagă între mai multe versiuni de kernel ce vor fi încărcate, selectând astfel între mai multe sisteme de operare. De asemenea, *bootloader*-ul permite și transmiterea unor opțiuni de încărcare a nucleului sistemului de operare. *Bootloader*-ul este încărcat în memorie de BIOS.

Bootloader-ul se regăsește pe primul sector al unui dispozitiv bootabil, denumit și **sectorul de boot**. Aceasta limitare de dimensiune apare deoarece BIOS-ul poate încărca în memorie un singur sector (512 octeți). Se poate întâmpla ca *bootloader*-ul să fie prea mare și să nu încapă în primul sector al dispozitivului boot-abil. În acest caz

Figura 6.3: BIOS POST (Power-On Self Test)

bootloader-ul se împarte în două componente. Prima componentă începe în sectorul de boot. A doua, *second stage*, rezidentă pe un spațiu mai mare de pe hard-disk și este încărcată de către prima. Un astfel de multiple-stage bootloader este GRUB, bootloader-ul implicit pe majoritatea distribuțiilor Linux.

6.2.1 Dispozitive boot-abile

Un dispozitiv boot-abil este un dispozitiv al cărui prim sector este un **sector boot-abil**. Sectorul boot-abil se caracterizează prin faptul că ultimii doi octeți sunt `0xAA55` (din cei 512).

Exemple de dispozitive ce pot fi boot-abile sunt CD-ROM-uri, hard-disk-uri, floppy disk-uri, USB flash etc. În cazul acestor dispozitive, dacă primul sector are structura specifică unui sector boot-abil atunci poate fi folosit pentru încărcarea altor informații existente.

După rularea POST, BIOS-ul consultă CMOS și urmează ordinea de boot-are de acolo. Astfel, dacă CMOS-ul a fost configurat ca în imaginea de mai sus, BIOS-ul va efectua următoarele operații:

- dacă HDD-ul are un prim sector bootabil, se va încărca și se va executa conținutul sectorului; aici se va afla, de obicei, bootloader-ul; în urma execuției se va încărca sistemul de operare;
- dacă HDD-ul nu are un prin sector boot-abil, se trece la investigația CD-ROM-ului din sistem;
- se va verifica existența unui CD în unitatea de CD-ROM; dacă acesta există, se va verifica existența sectorului de boot;

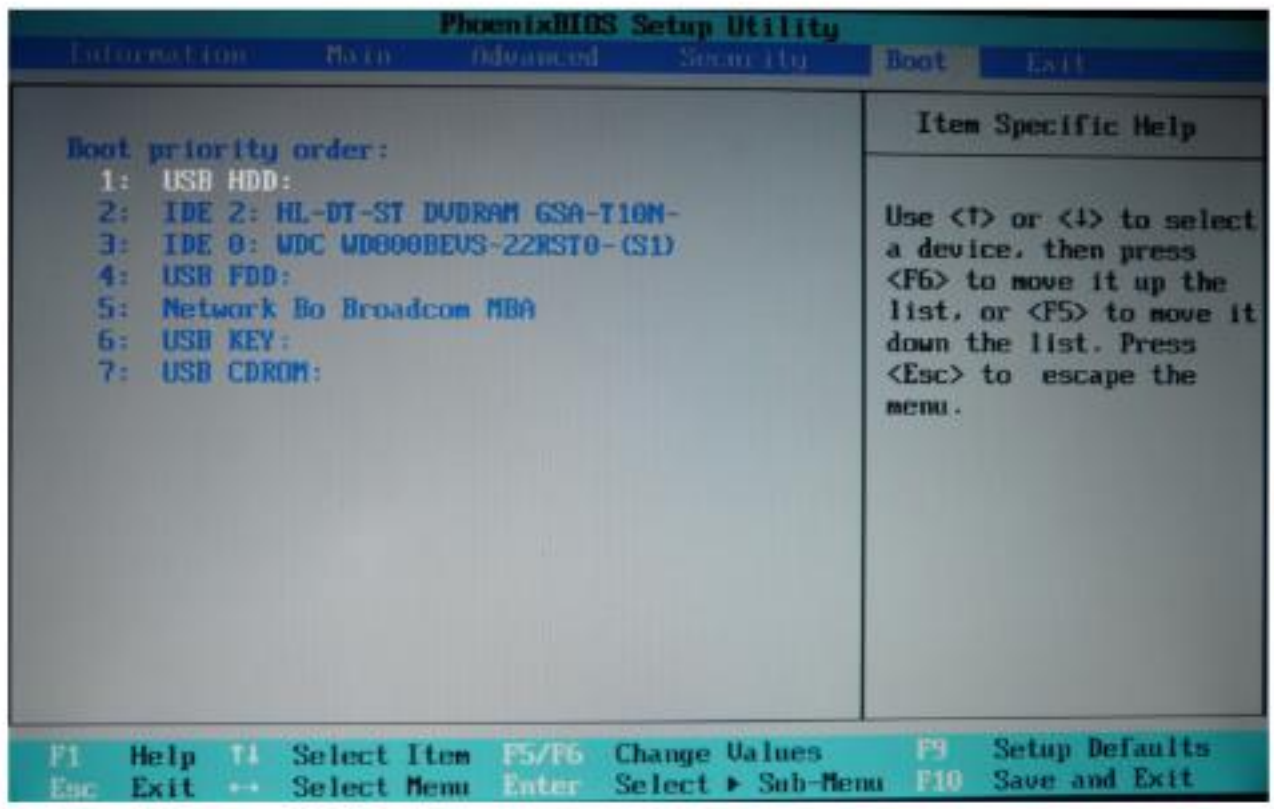


Figura 6.4: Alegerea dispozitivului de boot

- dacă CD-ul are un prim sector bootabil, atunci se va încărca și se va executa conținutul sectorului de boot; exemple de CD boot-abil sunt CD-ul de instalare a unui sistem de operare sau un LiveCD;
- dacă nici CD-ROM-ul nu are un prim sector bootabil atunci se trece la următorul dispozitiv boot-abil (al doilea HDD), care va fi analizat asemenea celor aterioare;
- dacă nu există niciun dispozitiv boot-abil, nu se poate încărca niciun sistem de operare; BIOS-ul va afișa un mesaj specific fără a executa nimic.

6.2.2 Structura sectorului de boot pentru un dispozitiv boot-abil

Primul sector (primii 512 octeți) al unui dispozitiv boot-abil poartă denumirea de **MBR** (*Master Boot Record*). Primul sector al unei partiții al unui dispozitiv boot-abil partiționat poartă numele de **VBR** (*Volume Boot Record*). După cum a fost specificat anterior, un dispozitiv este boot-abil dacă MBR-ul acestuia este un sector boot-abil (are ultimii doi octeți `0xAA55`, pentru a semnifica faptul că sectorul conține secvențe de cod executabile).

Sectorul de boot poate conține instrucțiuni pentru încărcarea unei secvențe mai mari de cod (cum este sectorul de boot al unui CD), sau poate conține un bootloader pentru încărcarea sistemului de operare (cum este sectorul de boot al unui hard-disk).

Conținutul sectorului de boot este încărcat în memorie de BIOS. După aceasta, BIOS-ul transmite controlul programului rezident în sectorul de boot (*bootloader*).

În figura 6.5 este prezentată structura MBR.

După cum se observă, există următoarele zone importante:

- **zona de cod executabil**; aceasta conține programul căruia îi este transmis controlul de către BIOS (*bootloader*);

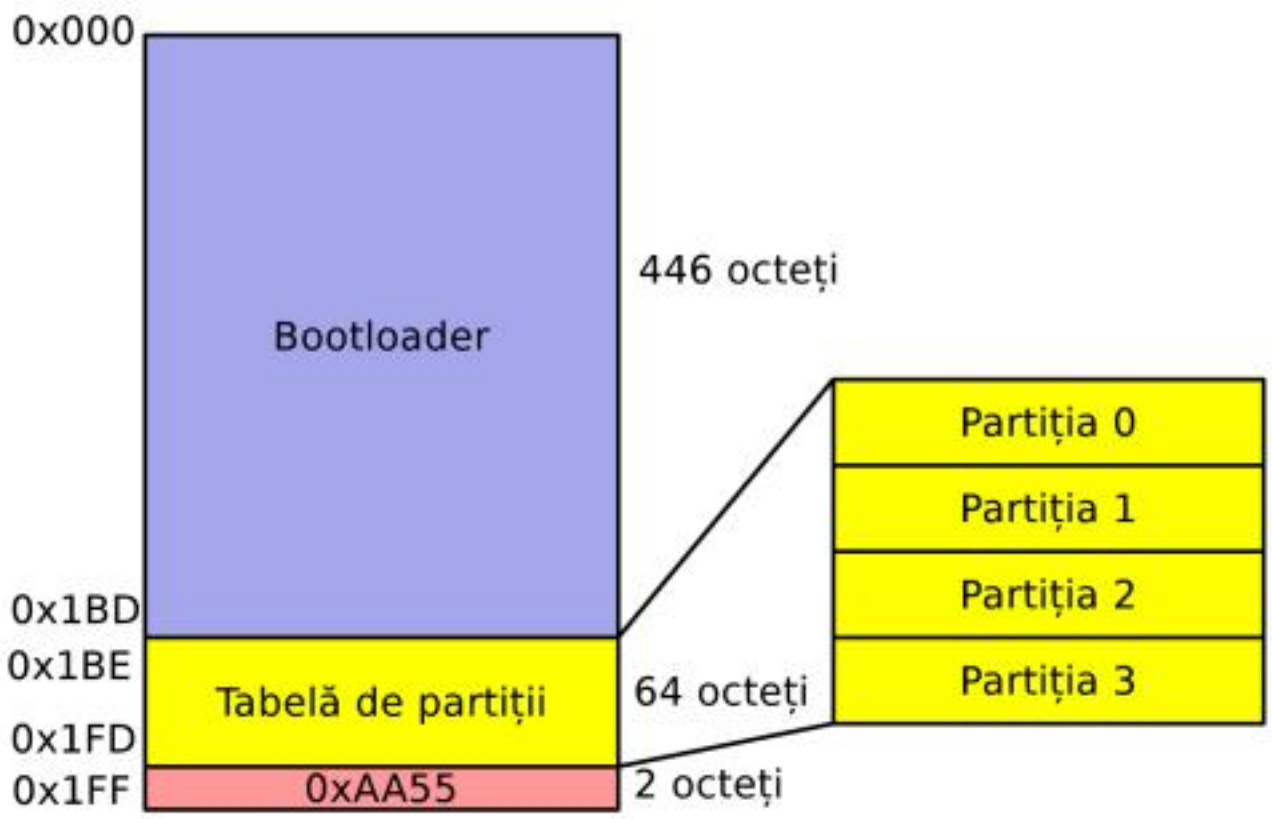


Figura 6.5: Structura MBR (Master Boot Record)

- **tabela de partiții** conține informații despre partițiile din sistem; fiecărei partiții îi corespunde o intrare de 16 octeți;
- **semnătura** de sector boot-abil (`0xAA55`).

Zona de cod executabil conține programul care va fi rulat după încărcarea MBR-ului în memorie de BIOS. De obicei, acest program este *bootloader*-ul, responsabil cu descoperirea partiției active și încărcarea primului sector (VBR) de pe aceasta. În cazul existenței mai multor sisteme de operare, *bootloader*-ul poate permite utilizatorului să opteze pentru încărcarea unui sistem de operare aflat pe altă partiție decât cea de boot.

Tabela de partiții

Tabela de partiții rezidă în MBR și ocupă 64 de octeți. Un hard-disk poate conține doar patru partiții (așa numitele **partiții primare**), rezultând 16 octeți pentru descrierea fiecărei partiții.

Cei 16 octeți ai fiecărei partiții oferă, pe lângă adresa de start (exprimată în *cylinder-head-sector*[1]) și dimensiunea și tipul partiției. O partiție poate fi activă sau non-activă. O partiție activă este aleasă de *bootloader*-ul aflat în zona MBR (sau de către o parte a *bootloader*-ului, dacă acesta este în două stagii) pentru a continua încărcarea sistemului. De pe o partiție activă se va încărca primul sector (VBR) al acesteia.

O altă clasificare a partițiilor este clasificarea în partiții primare și partiții extinse. Numărul de 4 **partiții primare** care pot fi descrise în tabela de partiții din MBR este, de multe ori, insuficient. Pentru a rezolva acest neajuns, a apărut noțiunea de **partiție extinsă**. Pe un sistem poate exista o singură partiție extinsă astfel că putem avea fie maxim 4 partiții primare, fie o partiție extinsă și maxim 3 primare.

[1] http://en.wikipedia.org/wiki/Cylinder-head-sector

Partițiile extinse pot conține mai multe **partiții logice**, mărind astfel numărul de partiții posibile din sistem. Descrierea partiției extinse se regăsește în primul sector din acea partiție, denumit **EPBR** (*Extended Partition Boot Record*).

Partițiile sunt folosite pentru separația informațiilor într-un sistem de operare. Pentru a asigura existența unui sistem de fișiere pe o partiție, acea partiție va trebui formatată (vezi secțiunea 4.8.1).

Salvarea sectorului de boot (backup)

Uneori se dorește salvarea sectorului de boot al unui disc sau al unei partiții pentru situația în care acesta devine corupt și trebuie reparat. În Linux, acest lucru se poate realiza foarte ușor prin intermediul comenzii **dd** (vezi secțiunea 7.6.5) și a dispozitivelor de tip bloc (vezi secțiunea 7.6.1)

Astfel, dacă se dorește realizarea unei copii locale a MBR, se poate folosi utilitarul **dd** și dispozitivul asociat hard-disk-ului (în cazul nostru `/dev/hda`). Vor trebui copiați 512 octeți (un sector), deci contorul de sectoare va fi 1:

```
1 root@asgard:~# dd if=/dev/hda of=mbr.out count=1
2 1+0 records in
3 1+0 records out
4 512 bytes (512 B) copied, 0.035273 seconds, 14.5 kB/s
5
6 root@asgard:~# ls -l mbr.out
7 -rw-r--r-- 1 root root 512 2009-08-07 23:28 mbr.out
```

6.2.3 Mecanismul de funcționare a unui bootloader

După cum s-a specificat, un *bootloader* este un program folosit pentru a încărca în RAM imaginea sistemului de operare, sau alte componente care vor încărca (la rândul lor) acea imagine.

Codul *bootloader*-ului este supus unor constrângeri destul de rigide, întrucât trebuie să încapă în MBR. Mai exact, este limitat la 446 de octeți (restul fiind ocupați de tabela de partiții și de semnătura de boot `0xAA55`).

Ca atare *bootloader*-ele lucrează în mod normal în două etape (*stages*). Prima etapă este cea care rezidă în MBR. Aceasta are rolul de a încărca de pe hard-disk a doua etapă care este suficient de complexă pentru a putea începe încărcarea efectivă a kernelului sistemului de operare. În cea de-a doua etapă poate fi oferit un ecran de selecție pentru situația în care coexistă două sisteme de operare pe același sistem fizic. Unele *bootloader*-e (spre exemplu GRUB versiunea 1) au o fază intermediară – 1.5 – pentru a permitea citirea de informații de pe hard-disk dincolo de o anumită limită.

După acest pas bootloader-ul va încărca nucleului sistemului de operare (sau o parte a acestuia) și va transmite controlul către acesta.

Exemple de bootloader-e sunt:

- **GRUB** (*GRand Unified Bootloader*): *bootloader*-ul implicit pe majoritatea distribuțiilor Linux;

- **LILO** (*Linux Loader*): *bootloader*-ul implicit pe distribuțiile Linux înainte de supremația GRUB;
- **NTLDR** (*NT Loader*): *bootloader*-ul folosit de Windows NT, 2000, XP, 2003;
- **winload.exe**: *bootloader*-ul folosit de Windows Vista;
- **SYSLINUX**: o suită de *bootloader*-e destinate diverselor medii de boot.

Chainloading

Modul cel mai des întâlnit prin care un sistem de operare este inițializat este prin specificarea directă a imaginii nucleului către *bootloader*.

Alternativa o reprezintă boot-area indirectă a sistemului de operare. Acest lucru presupune ca *bootloader*-ul aflat în MBR, în loc de imaginea unui kernel, să încarce un alt *bootloader* care va încărca la rândul său imaginea de kernel a sistemului de operare. Acest mod de încarcare a unui sistem de operare se numește **chainloading**.

Bootloader-ele specifice Linux (precum GRUB sau LILO) încarcă sistemele Windows folosind *chainloading*. Spre exemplu, dacă după încărcarea *bootloader*-ului GRUB se optează pentru boot-area Windows, atunci se va incărca *bootloader*-ul Windows (NTLDR) care va inițializa la rândul său sistemul de operare.

6.2.4 GRUB

GRUB (*GRand Unified Booloader*[1]) este bootloader-ul implicit pe distribuțiile Linux și alte sisteme de operare din familia Unix și este, de obicei, folosit în situațiile de **multiboot** (când pe sistemul de calcul coexistă mai multe sisteme de operare). Această secțiune va prezenta detalii despre versiunea 1 a GRUB, denumită simplu GRUB. Versiunea 2, denumită GRUB 2, care a început să fie utilizată recent, este prezentată în cadrul unui studiu de caz (vezi secțiunea 6.6.1).

Specific GRUB este faptul că există o fază intermediară 1.5 pentru situația în care există o limită de vizibilitate a hard-disk-ului.

Ca și în cazul altor *bootloader*-e, prima fază (cea care rezidă în MBR) are rolul de a încărca faza a doua sau, dacă este nevoie, faza 1.5. Datorită dimensiunii reduse, prima fază nu deține informații despre sistemul de fișiere. în schimb, faza 1.5 deține informații despre acesta și poate localiza și încărca în mod corespunzător faza a doua.

Faza a doua este responsabilă cu citirea fișierului principal de configurare (`/boot/grub/menu.lst`), afișarea ecranului de opțiuni și încărcarea nucleului sistemului de operare.

În figura 6.6 este prezentată localizarea diferitelor faze ale GRUB pe hard-disk.

Se observă că prima etapă (*grub stage 1*) se găsește în zona de cod executabil din MBR. Faza 1.5 (*grub stage 1.5*) se găsește în zona de compatibilitate a hard-disk-ului (primii 32 KB). Această etapă recunoaște tipul sistemului de fișiere pe care se află etapa

[1] http://www.gnu.org/software/grub/

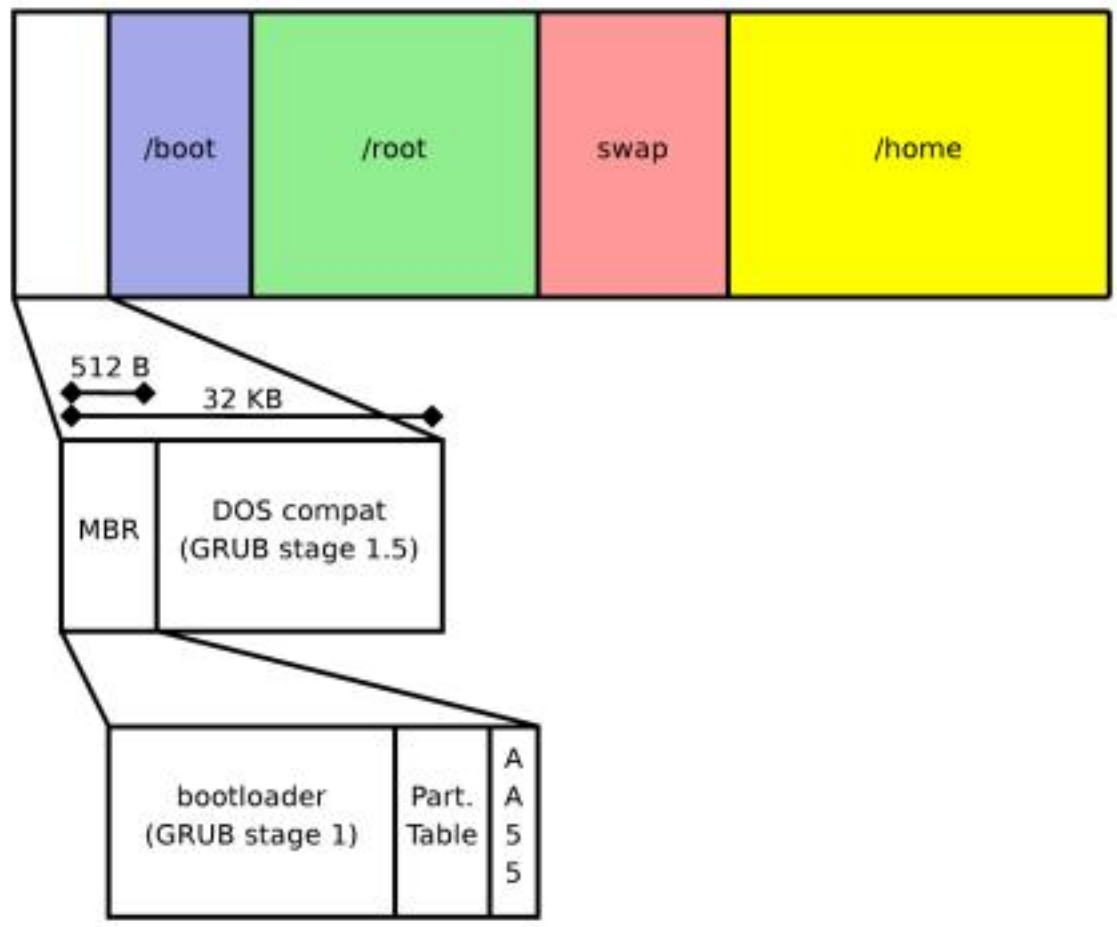


Figura 6.6: Structura GRUB

a doua și este utilizată pentru încărcarea acesteia. Etapa a doua rezidă în partiția de boot.

Fișierele asociate GRUB se găsesc pe un sistem Debian/Ubuntu, în `/boot/grub`, așa cum este prezentat mai jos:

```
1  razvan@asgard:/boot/grub$ ls
2  default         fat_stage1_5  minix_stage1_5     stage2
3  device.map      jfs_stage1_5  reiserfs_stage1_5  xfs_stage1_5
4  e2fs_stage1_5   menu.lst      stage1
5
6  razvan@asgard:/boot/grub$ ls -l
7  total 188
8  -rw-r--r-- 1 root root    197 2009-01-17 11:55 default
9  -rw-r--r-- 1 root root     15 2009-01-17 11:55 device.map
10 -rw-r--r-- 1 root root   7584 2009-01-17 11:55 e2fs_stage1_5
11 -rw-r--r-- 1 root root   7424 2009-01-17 11:55 fat_stage1_5
12 -rw-r--r-- 1 root root   8192 2009-01-17 11:55 jfs_stage1_5
13 -rw-r--r-- 1 root root   4724 2009-06-23 12:45 menu.lst
14 -rw-r--r-- 1 root root   6848 2009-01-17 11:55 minix_stage1_5
15 -rw-r--r-- 1 root root   9280 2009-01-17 11:55 reiserfs_stage1_5
16 -rw-r--r-- 1 root root    512 2009-01-17 11:55 stage1
17 -rw-r--r-- 1 root root 108392 2009-01-17 11:55 stage2
18 -rw-r--r-- 1 root root   8904 2009-01-17 11:55 xfs_stage1_5
```

Se observă că prima fază se găsește în fișierul `stage1` care are exact dimensiunea unui sector (512 octeți). Faza 1.5 are o formă specializată pentru fiecare sistem de fișiere (`e2fs_stage1_5`, `reiserfs_stage1_5` etc.). Spațiul ocupat este în jur de 7-8 KB, deci încape fără probleme în zona de compatibilitate de 32 KB a hard-disk-ului.

Faza a doua este descrisă de fișierul `stage2` cu dimensiune mai mare decât a celorlalte (circa 100 KB). Faza a doua este resposabilă pentru citirea fișierului de configurare `menu.lst` și încărcarea ulterioară a nucleului.

Vizualizare stage1 și stage1.5 pe hard-disk

Pentru a verifica prezența primei faze și a fazei 1.5 pe hard-disk vom folosi utilitarele **dd** și **hexdump**.

Într-o primă fază va fi realizată o copie a primului sector de pe hard-disk în fișierul `first_sector` și apoi a următoarelor 10 sectoare în fișierul `next_sectors`.

```
root@asgard:/boot/grub# dd if=/dev/hda of=first_sector count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 8.91e-05 seconds, 5.7 MB/s

root@asgard:/boot/grub# dd if=/dev/hda of=next_sectors count=10 skip=1
10+0 records in
10+0 records out
5120 bytes (5.1 kB) copied, 0.000137291 seconds, 37.3 MB/s
```

În `first_sector` și `next_sector` se găsesc informațiile binare conținute în primele sectoare ale hard-disk-ului. Ele vor fi comparate cu fișierele imagine din `/boot/grub` folosind utilitarul **hexdump**. De exemplu, pentru `stage1`:

```
razvan@asgard:/boot/grub$ hexdump first_sector
[...]
0000070 34e8 f601 80c2 5474 41b4 aabb cd55 5a13
0000080 7252 8149 55fb 75aa a043 7c41 c084 0575
0000090 e183 7401 6637 4c8b be10 7c05 44c6 01ff
00000a0 8b66 441e c77c 1004 c700 0244 0001 8966
[...]
0000170 be06 7d94 30e8 be00 7d99 2ae8 eb00 47fe
0000180 5552 2042 4700 6f65 006d 6148 6472 4420
0000190 7369 006b 6552 6461 2000 7245 6f72 0072
00001a0 01bb b400 cd0e ac10 003c f475 00c3 0000
[...]

razvan@asgard:/boot/grub$ hexdump stage1
[...]
0000070 34e8 f601 80c2 5474 41b4 aabb cd55 5a13
0000080 7252 8149 55fb 75aa a043 7c41 c084 0575
0000090 e183 7401 6637 4c8b be10 7c05 44c6 01ff
00000a0 8b66 441e c77c 1004 c700 0244 0001 8966
[...]
0000170 be06 7d94 30e8 be00 7d99 2ae8 eb00 47fe
0000180 5552 2042 4700 6f65 006d 6148 6472 4420
0000190 7369 006b 6552 6461 2000 7245 6f72 0072
00001a0 01bb b400 cd0e ac10 003c f475 00c3 0000
[...]
```

După cum se observă, în primul sector al hard-disk-ului se găsește imaginea primei etape a GRUB, iar în următoarele sectoare imaginea etapei 1.5. Dacă apar deosebiri între imaginea aflată în sectoarele discului și cea din `/boot/grub`, acestea sunt din considerente de adaptare la geometria discului. Imaginea celei de-a doua faze este încărcată de faza 1.5 din `/boot/grub/stage2`, aceasta putând accesa structura sistemului de fișiere.

Creare unei dischete de boot GRUB

Deși dischetele sunt dispozitive pe cale de dispariție (și costă în ziua de azi mai mult decât un CD), vom descrie în scop didactic modul în care se poate crea o dischetă de boot GRUB.

Discheta va trebui să conțină imaginea pentru prima și cea de-a doua fază (nu este nevoie de faza 1.5 întrucât nu avem limitare în cest caz). Pentru aceasta se poate folosi folosi utilitarul **cat**:

```
root@anaconda:/boot/grub# cat stage1 stage2 > /dev/fd0
```

sau utilitarul **dd**:

```
root@anaconda:/boot/grub# dd if=stage1 of=/dev/fd0
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.0426716 seconds, 12.0 kB/s

root@anaconda:/boot/grub# dd if=stage2 of=/dev/fd0 skip=1
210+1 records in
210+1 records out
107656 bytes (108 kB) copied, 0.020834 seconds, 5.2 MB/s
```

Dacă se configurează BIOS-ul pentru a boot-a de pe dischetă, utilizatorului îi va fi oferită consola GRUB de configurare (vezi secțiunea 6.3).

6.3 Configurarea GRUB

Configurarea GRUB se realizează prin intermediul fișierului `/boot/grub/menu.lst`. În acest fișier sunt precizate opțiunile de boot prezente în meniul GRUB, cu alte cuvinte sistemele de operare care pot fi încărcate cu ajutorul GRUB. Etapa a doua citește acest fișier de configurare și oferă utilizatorului ecranul de opțiuni:

În continuare este prezentată o parte dintr-un fișier de configurare `/boot/grub/menu.lst` și sunt descrise elementele de configurare prezente în acesta:

```
## default num
# Set the default entry to the entry number NUM. Numbering starts from 0,
 and
# the entry number 0 is the default if the command is not used.
[...]
default         0

## timeout sec
# Set a timeout, in SEC seconds, before automatically booting the default
 entry
# (normally the first entry defined).

timeout         5

# Pretty colours
color cyan/blue white/blue
[...]
```

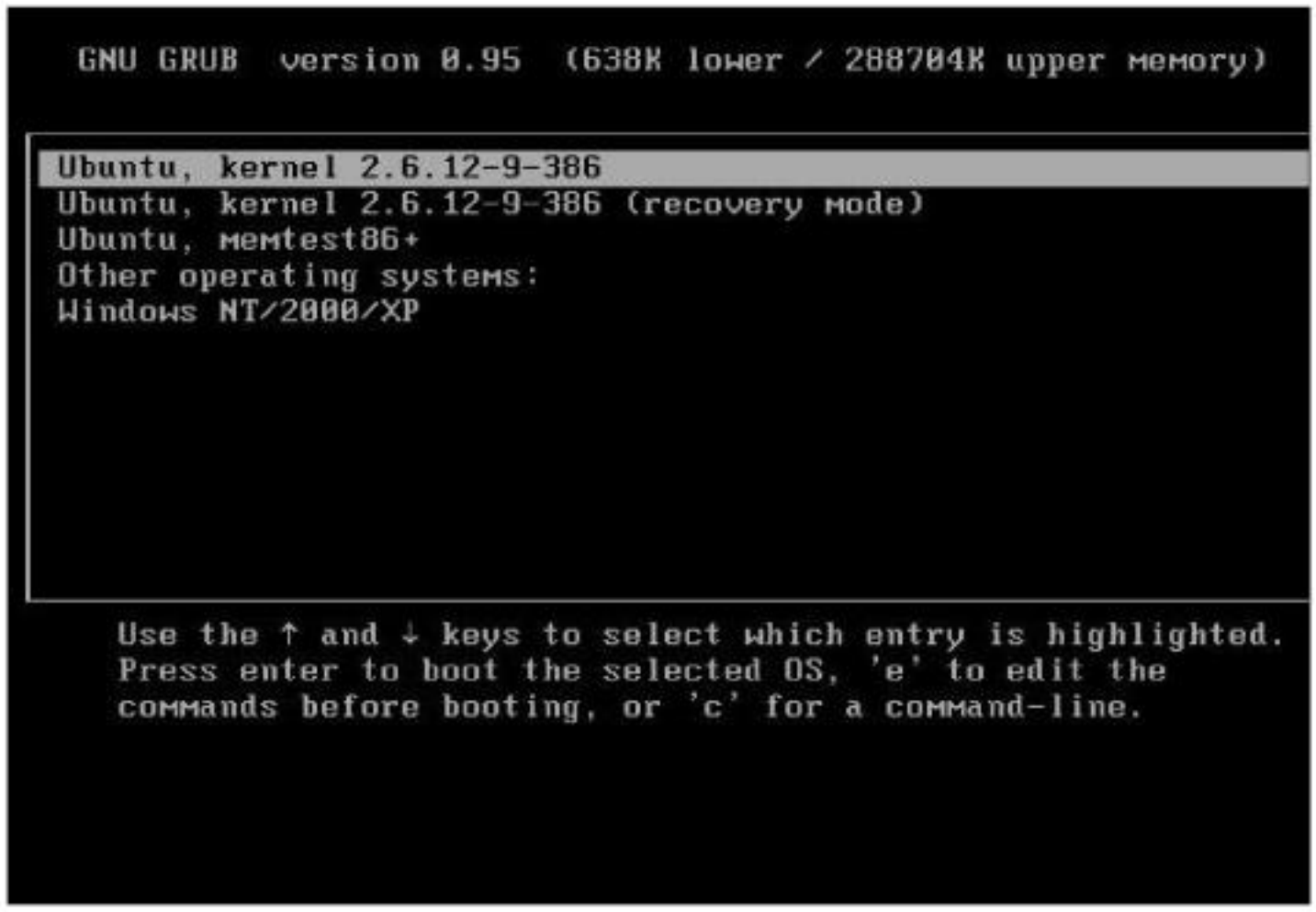


Figura 6.7: Ecran opțiuni GRUB

```
16
17 ## Razvan's kernels
18
19 title           Debian GNU/Linux, kernel 2.6.18-12fast
20 root            (hd0,2)
21 kernel          /boot/vmlinuz-2.6.18-12fast root=/dev/hda3 ro
22 savedefault
23
24 title           Debian GNU/Linux, kernel 2.6.18-12fast (single-user mode)
25 root            (hd0,2)
26 kernel          /boot/vmlinuz-2.6.16-12fast root=/dev/hda3 ro single
27 savedefault
28 [...]
29
30 # This entry automatically added by the Debian installer for a non-linux
   OS
31 # on /dev/hda1
32 title           Microsoft Windows XP Professional
33 root            (hd0,0)
34 savedefault
35 makeactive
36 chainloader     +1
```

Opțiunea `default` precizează opțiunea implicită din meniu folosită pentru boot-are. Fișierul de configurare conține mai multe intrări pentru diferite sisteme de operare. Numărul care urmează după opțiunea `default` indică sistemul de operare care va fi boot-at implicit în cazul în care utilizatorul nu intervine în momentul afișarii meniului de opțiuni. Numerotarea pornește de la 0. Presupunând că, în figura 6.7, nu ar exista alte opțiuni de sisteme de operare, boot-area implicită pe Windows XP Professional este condiționată de prezența opțiunii default 4.

Opțiunea `timeout` indică timpul de așteptare pentru ca utilizatorul să aleagă una din

opțiunile afișate în meniul de boot. Dacă acest timp expiră, se va încărca sistemul de operare indicat de opțiunea `default`.

Opțiunea `colors` afișează colorat meniul GRUB.

Opțiunea de sistem de operare

Ceea ce apare, în continuare, în exemplul de fișier de configurare de mai sus sunt intrările asociate pentru sistemele de operare care pot fi boot-ate. Se observă că fiecare intrare conține un set de directive de configurare. Acestea sunt:

- `title`: este un șir de caractere reprezentând numele intrării așa cum va apărea ea în ecranul de opțiuni GRUB. Deși nu există nicio limitare de nume, se recomandă ca titlul să fie un nume reprezentativ;
- `root`: reprezintă partiția pe care se află imaginea de nucleu a sistemului de operare; aceasta poate sau nu coincidă cu partiția unde este montat sistemul de fișiere rădăcină (`/`);
- `kernel`: reprezintă calea către imaginia de nucleu în cadrul partiției de mai sus; subdirectiva `root` precizează partiția asociată sistemului de fișiere rădăcină (`/`); opțiunea `ro` precizează faptul că sistemul de fișiere va fi montat read-only; se observă că în cazul Windows nu există opțiunea `kernel`, deorece GRUB nu poate boot-a direct nucleul Windows ci va apela NTLDR (*bootloader*-ul de Windows), lucru indicat prin opțiunea `chainloader+1`; opțiunea `+1` semnifică citirea primului sector din partiția `(hd0,0)`;
- opțiunea `makeactive` stabilește partiția ca fiind partiția activă;
- opțiunea `savedefault`: dacă directiva `default` este folosită în forma `default saved` și opțiunea folosită la boot-are are opțiunea `savedefault`, atunci la următoarea boot-are se va selecta, implicit, aceeași opțiune.

Convenția de denumire GRUB

Dacă în Linux, primul hard-disk este accesibil sub numele `/dev/hda`, iar prima partiție a acestuia sub numele `/dev/hda1`, GRUB folosește o altă convenție pentru denumirea discurilor din sistem și a partițiilor acestora. Astfel, pentru a referi primul disc din sistem, GRUB folosește denumirea `(hd0)`. Numerotarea discurilor începe de la 0. Spre deosebire de Linux, GRUB nu face distincție între tipul de controller folosit de discurile din sistem: IDE sau SCSI. Astfel, primul disc este denumit întotdeauna `(hd0)`. Pentru referirea primei partiții din primul disc, GRUB folosește denumirea `(hd0,0)`.

În ton cu cele de mai sus, a 3-a partiție de pe al doilea disc al sistemului este referită de GRUB sub forma `(hd1,2)`. Tabelul 6.1 de mai sus oferă o analogie între denumirea GRUB și Linux pentru discuri și partiții.

Consola GRUB

În afara fișierului de configurare, GRUB poate fi configurat și prin intermediul unei consolei asociate. Aceasta poate fi accesată în două moduri: dintr-un sistem Linux

Tabelul 6.1: Denumirea discurilor în Linux/GRUB

Dispozitiv	**Denumire Linux**	**Denumire GRUB**
Primul disc IDE	`/dev/hda`	`(hd0)`
Al doilea disc IDE	`/dev/hdb`	`(hd1)`
Al treilea disc SCSI	`/dev/sdc`	`(hd2)`
Prima unitate floppy	`/dev/fd0`	`(fd0)`
Prima partitie de pe al doilea disc IDE	`/dev/hdb1`	`(hd1,0)`
A doua partitie de pe al treilea disc SCSI	`/dev/sdc2`	`(hd2,1)`

folosind comanda **`grub`**, sau din intermediul meniului de boot al GRUB. În ambele cazuri utilizatorului îi este oferit un prompt `grub>` unde poate introduce comenzi specifice.

Utilizarea consolei GRUB este limitată la situațiile de testare a unor opțiuni, de recuperare sau de troubleshooting.

Un exemplu util de utilizare a consolei GRUB este (re)instalarea GRUB, așa cum este descrisă în secțiunea 6.6.3.

Accesarea consolei GRUB poate fi necesară și în momentul în care se dorește alterarea unei opțiuni de boot-are. Spre exemplu, în ecranul de meniul de boot GRUB, apăsarea tastei `e` peste o anumită opțiune conduce la apariția consolei GRUB cu posibilitatea editării acelei opțiuni de bootare. Acest lucru este util în cazul unei erori (s-a precizat de exemplu un nume greșit pentru imaginea de kernel) sau în cazul în care s-a pierdut parola de root și se dorește recuperarea acesteia, forțând sistemul să pornească o consolă de root (prin folosirea opțiunii `init=/bin/bash` (vezi secțiunea 10.6.2).

6.4 Încărcarea nucleului

După ce este încărcat în RAM și rulat, *bootloader*-ul este responsabil cu încărcarea nucleului sistemului de operare și transmiterea controlului către acesta. Ca și faza a doua a GRUB, nucleul rezidă într-un fișier din sistemul de fișiere, denumit **imaginea de nucleu**. În cazul GRUB, locația acestui fișier este transmisă ca parametru în cadrul directivei `kernel` din `/boot/grub/menu.lst`.

6.4.1 Imaginea de nucleu

Imaginea de nucleu este un fișier executabil conținând codul și datele necesare pentru rularea nucleului sistemului de operare. În Windows, imaginea nucleului se numește `ntoskrnl.exe` și este localizată în `C:\Windows\system32\ntoskrnl.exe`. În sistemele Linux, această imagine rezidă, de obicei, în `/boot`, și are denumirea în forma `vmlinuz-versiune_kernel` (spre exemplu, `vmlinuz-2.6.29.1` sau `vmlinuz-2.6.31-020631-generic`).

Comanda **`file`** indică un astfel de fișier ca fiind o imagine de nucleu Linux:

```
1 root@asgard:/boot# file vmlinuz-2.6.31-020631-generic
2 vmlinuz-2.6.31-020631-generic: Linux kernel x86 boot executable RO-rootFS
  , root_dev 0x6801, swap_dev 0x3, Normal VGA
```

În mod obișnuit imaginea de kernel ocupă în jur de 3-4 MB. Poate fi însă constrânsă la dimensiuni mai mici prin eliminarea anumitor componente, atunci când este cazul (spre exemplu în cazul sistemelor *embedded*[1]). Faptul că denumirea imaginii se termină în `z` (`vmlinuz`) indică o imagine de kernel comprimată.

O imagine de kernel comprimată conține la începutul ei o secțiune direct executabilă, și după această secțiune, se găsește zona efectiv comprimată.

După încărcarea imaginii de nucleu, *bootloader*-ul transmite controlul execuției către kernel (fie către imaginea necomprimată, fie către zona necomprimată). Într-o primă fază nucleul inițializează și configurează dispozitivele hardware și memoria. Toate dispozitivele inițializate de BIOS vor fi reinițializate pentru a asigura robustețea și independența de modul în care BIOS-ul realizează initializarea.

Următorul pas îl reprezintă decomprimarea imaginii de nucleu. După aceasta se montează sistemul de fișiere rădăcină precizat prin parametrul `root=/dev/...` al directivei `kernel` din fișierul de configurare GRUB. Pe măsură ce nucleul inițializează alte subsisteme, pe ecran sunt afișate mesaje de status.

Încărcarea completă a nucleului se încheie cu crearea procesului `init` (ce are pid-ul 1) din imaginea `/sbin/init`. Se poate alege o altă imagine pe baza căreia să se creeze procesul `init`, prin specificarea opțiunii `init=/path/to/new/exec` la boot-area nucleului. Un caz tipic pentru aceasta este recuperarea parolei de root (vezi secțiunea 10.6.2).

6.4.2 Opțiuni de boot-are pentru nucleu

Nucleul Linux poate fi configurat pentru boot-are cu diverse argumente care să-i modifice funcționalitatea. Acestea sunt transmise ca parametri de boot-are în directiva `kernel` a GRUB. Configurarea acestor argumente poate fi realizată fie prin fișierul de configurare (`/boot/grub/menu.lst`), fie prin accesarea consolei GRUB (apăsarea tastei `e` în ecranul de opțiuni GRUB).

O listă completă cu opțiunile de boot-are poate fi găsită online[2]. Cele mai importante dintre aceste opțiuni sunt:

- **debug**: afișează mesaje suplimentare la încărcarea nucleului;
- **quiet**: dezactivează afișarea de mesaje la încărcarea nucleului;
- **resume=suspend_device** (util pentru laptop-uri în cazul fincției de hibernare): specifică discul unde se află imaginea suspendată a nucleului;
- **root=device**: precizează partiția unde se află sistemul de fișiere rădăcină;

[1] http://en.wikipedia.org/wiki/Embedded_system
[2] http://www.kernel.org/pub/linux/kernel/people/gregkh/lkn/lkn_pdf/ch09.pdf

- **ro/rw**: specifică dacă sistemul de fișiere rădăcină este montat *read-only* sau *read-write* la boot-are;
- **init=filename**: specifică executabilul care va fi încărcat la crearea procesului `init` (implicit acest executabil este `/sbin/init`);
- **S** sau **single**: specifică pornirea lui `init` în mod *single-user*.

6.5 Inițializarea sistemului

Încărcarea nucleului se încheie cu pornirea `init`, procesul cu pid-ul 1, părintele celorlalte procese din sistem. În cele ce urmează vom presupune că procesul init a fost creat în mod obișnuit din imaginea `/sbin/init`.

`init` este responsabil cu pornirea celorlalte procese importante ale sistemului, a daemonilor și a proceselor care asigură accesul utilizatorului la sistem (`getty` și mediul grafic).

6.5.1 init

`init` este resposabil cu crearea celorlalte procese importante din sistem și, astfel, de realizarea unui sistem funcțional și interactiv. Pentru precizarea proceselor care trebuiesc pornite și a opțiunilor primite de către acestea, `init` folosește un set de fișiere de configurare decrise în continuare.

Niveluri de rulare (runlevels)

Înainte de a prezenta modul în care se configurează `init`, trebuie descrisă noțiunea de **nivel de rulare** (*runlevel*). Un nivel de rulare se referă la un context de utilizare a unui sistem de operare. Altfel spus, se referă la ce servicii oferă sistemul utilizatorului. Un *runlevel* diferă de un altul prin dispozitivele montate și prin serviciilor pornite.

Sunt definite 7 niveluri de rulare, numerotate de la 0 la 6. Acestea pot fi folosite diferit de diversele distribuții Linux, însă 3 dintre ele au tot timpul aceeași semnificație:

- **Runlevel 0** – *Halt*: este folosit pentru oprirea sistemului;
- **Runlevel 1** – `Single`: este folosit pentru un sistem single-user;
- **Runlevel 6** – `Reboot`: este folosit pentru repornirea sistemului.

Pentru afișarea *runlevel*-ului curent se folosește comanda **`runlevel`** (ca root) sau **`who -r`** (ca utilizator obisnuit):

```
1 root@asgard:~# runlevel
2 N 2
3
4 razvan@asgard:~$ who -r
5          run-level 2  2009-09-23 17:57                   last=
```

Schimbarea runlevel-ului se realizează prin rularea comenzii **`init`** urmată de numărul runlevel-ului. Astfel rularea comenzii

```
root@asgard:~# init 0
```

este echivalentă cu rularea comenzii:

```
root@asgard:~# shutdown -h
```

sau

```
root@asgard:~# halt
```

Configurarea init

Fișierul principal de configurare pentru `init` este `/etc/inittab`. Este prezentată, în continuare, o parte a unui fișier de configurare `/etc/inittab` pentru a descrie modul în care acesta afectează comportamentul `init`:

```
[...]
# The default runlevel.
id:2:initdefault:
[...]

# /etc/init.d executes the S and K scripts upon change
# of runlevel.
#
# Runlevel 0 is halt.
# Runlevel 1 is single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.

l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
# Normally not reached, but fallthrough in case of emergency.
z6:6:respawn:/sbin/sulogin

# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
[...]
# /sbin/getty invocations for the runlevels.
#
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
#
# Format:
#  <id>:<runlevels>:<action>:<process>
#
# Note that on most Debian systems tty7 is used by the X Window System,
# so if you want to add more getty's go ahead but skip tty7 if you run X.
#
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
```

```
40 3:23:respawn:/sbin/getty 38400 tty3
41 4:23:respawn:/sbin/getty 38400 tty4
42 5:23:respawn:/sbin/getty 38400 tty5
43 6:23:respawn:/sbin/getty 38400 tty6
```

După cum se poate observa intrările din `/etc/inittab` au următorul format:

```
1 <id>:<runlevels>:<action>:<process>
```

unde:

- `id` este un identificator unic al regulii din inittab;
- `runlevels` reprezintă nivelurile de rulare asupra cărora se aplică regula;
- `action` este acțiunea pe care o va executa init;
- `process` este comanda care va fi executată.

O descriere completă a regulilor posibile pentru `inittab` se găsește în pagina de manual (**`man inittab`**). Vor fi prezentate, în continuare, regulile din exemplul de mai sus.

Linia 3 din captura de mai sus stabilește *runlevel*-ul implicit, care este `2`.

Identificatorii `10` până la `16` de pe liniile 14-20 prezintă reguli pentru scripturile de inițializare a serviciilor sistemului conform cu *runlevel*-ul. După cum se observă, comanda executată pentru fiecare *runlevel* în parte este `/etc/init.d/rc` urmată de numărul nivelului de rulare. Prin intermediul acestei comenzi se vor porni serviciile specifice sistemului, după cum se va vedea în continuare.

Linia cu identificatorul `ca` (linia 25) precizează că se execută o repornire a sistemului în cazul în care se apasă `CTRL-ALT-DEL`. Pentru această repornire se rulează comanda **`shutdown`**.

Regulile cu identificatorii 1 până la 6 (liniile 38-43) sunt folosite pentru a porni procesul `getty` pe terminalele virtuale ale sistemului. După cum se observă și din comentariu, terminalul virtual 7 este asociat mediului grafic și nu poate fi folosit pentru a porni `getty`. Acțiunea `respawn` semnifică faptul că, în cazul terminării procesului, `init` îl va recrea.

Pornirea daemonilor de sistem

După cum s-a observat din fișierul de configurare `/etc/inittab`, pornirea serviciilor sistemului în funcție de *runlevel* se realizează cu ajutorul scriptului `/etc/init.d/rc`. Acest script primește ca argument numărul *runlevel*-ului și execută scripturile aflate într-un director specializat: `/etc/rcX.d/`, unde `X` este numărul *runlevel*-ului. Astfel, dacă *runlevel*-ul este 3, se vor executa script-urile din `/etc/rc3.d/`.

Conținutul directorului `/etc/rc3.d/` este:

```
1 razvan@asgard:~$ ls -l /etc/rc3.d/
2 total 4
3 lrwxrwxrwx 1 root   root    18 2009-06-10 22:56 K08vmware -> /etc/init.d/
  vmware
4 lrwxrwxrwx 1 root   root    20 2009-07-22 14:17 K77ntp-server -> ../init.
  d/ntp-server
5 [...]
```

```
6  lrwxrwxrwx 1 razvan razvan  15 2009-01-17 11:54 S20exim4 -> ../init.d/
   exim4
7  lrwxrwxrwx 1 razvan razvan  19 2009-06-01 12:51 S20firebird2 -> ../init.d
   /firebird2
8  [...]
9  lrwxrwxrwx 1 root   root    18 2009-06-10 22:56 S90vmware -> /etc/init.d/
   vmware
10 lrwxrwxrwx 1 razvan razvan  17 2009-02-24 00:05 S91apache2 -> ../init.d/
   apache2
11 lrwxrwxrwx 1 razvan razvan  18 2009-01-17 11:47 S99rc.local -> ../init.d/
   rc.local
12 lrwxrwxrwx 1 razvan razvan  19 2009-01-17 11:47 S99rmnologin -> ../init.d
   /rmnologin
13 lrwxrwxrwx 1 razvan razvan  23 2009-01-17 11:47 S99stop-bootlogd -> ../
   init.d/stop-bootlogd
```

Se observă că fișierele sunt un set de legaturi simbolice către intrări din `/etc/init.d/`. Directorul `/etc/init.d/` conține scripturile de pornire/oprire/repornire a serviciilor din sistem. Astfel, pentru oprirea, pornirea sau repornirea serviciului `apache`, se folosesc comenzile:

```
1 root@asgard:~# /etc/init.d/apache start|stop|restart
```

Argumentele `start` / `stop` / `restart` sunt comune majorității scripturilor ce se găsesc în `/etc/init.d`.

De asemenea, se poate observa și un format al numelor fișierelor din `/etc/rcX.d`:

```
1 K numar nume
2 S numar nume
```

`K` înseamnă *kill*, iar `S` înseamnă *start*. Astfel, existența legăturii `K08vmware` înseamnă că serviciul `vmware` va fi **oprit** (prin intermediul comenzii `/etc/init.d/vmware stop`) odată cu intrarea în nivelul de rulare 3. La fel, existența legăturii `S20exim4` înseamnă că serviciul `exim` va fi **pornit** (prin intermediul comenzii **`/etc/init.d/exim start`**) odată cu intrarea în nivelul de rulare 3.

Numărul asociat unui din formatul numelor de fișiere reprezintă ordinea de pornire/oprire a serviciilor. Astfel, serviciul `S91apache` va fi pornit înaintea `S99rmnologin`, iar serviciul `K77ntp-server` va fi oprit înaintea `K08vmware` (la oprire ordinea este descrescătoare).

sysv-rc-conf

Avantajul schemei `init` de pornire a serviciilor este flexibilitatea, iar dezavantajul – complexitatea crescută și un număr mare de fișiere de configurare. Dacă se dorește eliminarea sau adăugarea unui serviciu la un *runlevel*, va trebui creată o legătură simbolică către scriptul de pornire/oprire și denumită folosind schema specifică. Procesul poate fi anevoios în cazul în care se dorește o configurare mai avansată.

Pentru a elimina acest deficient, se poate folosi utilitarul **`sysv-rc-conf`**. Acesta permite adăugarea/eliminarea facilă a unui serviciu dintr-un runlevel.

`sysv-rc-conf` oferă două interfețe de configurare a serviciilor: una în linia de comandă și una text. Interfața text permite parcurgerea serviciilor existente în sistem și adăugarea/elimitarea acestora dintr-un *runlevel* printr-o simplă acțiune de

check/*uncheck* (folosind tasta `SPACE`). Interfața în linia de comandă permite adăugarea/eliminarea unui serviciu prin precizarea nivelului/nivelurilor de rulare dorite, a numelui serviciului și a acțiunii (*on* sau *off*). În exemplul de mai jos, serviciul `postfix` a fost adăugat la nivelurile de rulare 4 și 5 și serviciul `apache` a fost eliminat din nivelurile de rulare 2 și 5.

```
root@anaconda:/boot/grub# sysv-rc-conf --list postfix
postfix      0:off      1:off   2:on    3:off   4:off   5:off   6:off

root@anaconda:/boot/grub# sysv-rc-conf --level 45 postfix on

root@anaconda:/boot/grub# sysv-rc-conf --list postfix
postfix      0:off      1:off   2:on    3:off   4:on    5:on    6:off

root@anaconda:/boot/grub# sysv-rc-conf --list apache
apache       0:off      1:off   2:on    3:on    4:on    5:on    6:off

root@anaconda:/boot/grub# sysv-rc-conf --level 25 apache off

root@anaconda:/boot/grub# sysv-rc-conf --list apache
apache       0:off      1:off   2:off   3:on    4:on    5:off   6:off
```

6.5.2 upstart

Începând cu versiunea 6.10 (Edgy Eft), distribuția Ubuntu a înlocuit `init` cu **upstart**. upstart permite o configurare asincronă, bazată pe evenimente care au loc în sistem. Spre deosebire de alte alternative pentru `init`, upstart este compatibil cu acesta și poate folosi scripturile `init` nemodificate (spre exemplu cele din `/etc/init.d`). upstart este, de fapt, un daemon `init` modificat. Ca orice sistem Linux, upstart este pornit prin rularea `/sbin/init`.

În septembrie 2009 echipa de dezvoltare a distribuției Debian a anunțat faptul că va renunța la init în favoarea upstart[1].

Upstart a apărut ca urmare a dezvoltării hardware-ului. Dacă până de curând se putea ști din momentul pornirii sistemului ce dispozitive există în sistem sau unde se poate monta sistemul de fișiere, dezvoltarea de noi dispozitive *hot-plug* (care pot fi conectate sau deconectate în timp real în calculator) a impus folosirea unui nou sistem de detecție și configurare a acestora.

upstart este așadar un daemon init bazat pe evenimente (*event-based init daemon*). Funcționarea sa înseamnă crearea de noi sarcini pentru sistemul de operare în două moduri:

- rularea de scripturi sau executabile în momentul producerii unui eveniment;
- oprirea sau pornirea unui serviciu în momentul producerii unui eveniment.

Un exemplu este montarea unui sistem de fișiere (spre exemplu de pe un USB *flash drive*). Pașii urmați, după conectarea dispozitivului USB, sunt:

- un eveniment de tip *startup* pornește daemon-ul `udevd`;

[1] http://lwn.net/Articles/351013/

- daemon-ul `udevd` este configurat să trimită un eveniment de tipul *block-device-added* pentru fiecare dispozitiv nou conectat;
- evenimentul *block-device-added* determină verificarea dispozitivului și montarea acestuia dacă este listat în `/etc/fstab`;
- după montarea sistemelor de fișiere din `/etc/fstab` se transmite evenimentul *fhs-filesystem*.

Configurare upstart

upstart este configurat prin intermediul directorului `/etc/event.d/` care se dorește a fi un înlocuitor pentru scripturile `/etc/inittab`, `/etc/init.d/*` și `/etc/rcX.d`. Fiecare fișier din `/etc/event.d` descrie modul în care un job trebuie coordonat:

```
root@ubuntui:/etc/event.d# ls
control-alt-delete  rc0  rc3  rc6          sulogin  tty3  tty6
logd                rc1  rc4  rcS          tty1     tty4
rc-default          rc2  rc5  rcS-sulogin  tty2     tty5
```

Spre exemplu, fișierul `control-alt-delete` prezintă acțiunea întreprinsă la apăsarea `CTRL-ALT-DEL`, și anume repornirea sistemului:

```
root@ubuntui:/etc/event.d# cat control-alt-delete
# control-alt-delete - emergency keypress handling
#
# This task is run whenever the Control-Alt-Delete key combination is
# pressed.  Usually used to shut down the machine.

start on control-alt-delete

exec /sbin/shutdown -r now "Control-Alt-Delete pressed"
```

Scripturile `rc*` sunt aociate cu nivelurile de rulare. După cum se observă, la fel ca în `/etc/inittab`, se apelează `/etc/init.d/rc` cu argument numărul nivelului de rulare:

```
root@ubuntui:/etc/event.d# cat rc2
[..]
start on runlevel 2
stop on runlevel [!2]
script
[...]
        exec /etc/init.d/rc 2
end script
```

Scripturile `tty*` prezintă modul de pornire a getty pentru diversele terminale virtuale (la fel ca în `/etc/inittab`).

Începând cu Ubuntu 7.04 (Feisty Fawn), upstart a înlocuit câteva procese daemon importante cum sunt `cron`, `atd`, `anacron` și `inetd`.

6.5.3 getty și login

După pornirea daemonilor sistemului, `init` este responsabil cu pornirea proceselor care asigură autentificarea utilizatorului în sistemul de calcul. După cum s-a văzut în cadrul fișierului de configurare `/etc/inittab`, `init` pornește câte un proces getty pentru fiecare terminal virtual pe baza imaginii `/sbin/getty`. În mod implicit sunt pornite 6 procese getty pentru terminalele denumite `tty1`, `tty2`, ..., `tty6`. Terminalul `tty7` este rezervat mediului grafic (în cazul în care acesta este instalat). Autentificarea în mediul grafic (prin introducerea unui utilizator/parolă) nu se realizează de către getty, ci de *display manager* (vezi secțiunea 13.2.2), o componentă a sistemului de ferestre.

Terminalele virtuale sunt accesate prin intermediul combinației de taste `CTRL-ALT-Fn`, unde `n` este numărul terminalului. Combinația `CTRL-ALT-F7` este folosită pentru accesarea mediului grafic.

getty este responsabil cu crearea unei interfețe de autentificare a utilizatorului și este responsabil atât pentru autentificarea prin terminale virtuale cât și pentru conexiunea pe alte terminale precum interfețele seriale.

După inițializarea terminalului, getty oferă prompt-ul "`Login:`" către utilizator pe terminalul virtual. Utilizatorul va trebui să introducă numele de utilizator și să apese `ENTER`. După introducerea numelui de utilizator, getty invocă `/bin/login`. Astfel, imaginea sa este înlocuită cu imaginea `/bin/login` și se creează procesul `login`.

În continuare este cerută parola utilizatorului (ale cărei caractere nu vor fi afișate pe ecran în momentul tastării lor din motive de siguranță). În acest moment, `login` poate verifica validitatea utilizatorului care a dorit să se autentifice. Dacă utilizatorul a introdus o pereche nume de utilizator/parolă corectă atunci, în mod tipic, i se va deschide o sesiune de shell pe terminalul curent. Altfel, procesul login moare, și, după un interval de așteptare, getty este repornit și oferă din nou promptul de autentificare utilizatorului.

Schema de autentificare și pornire a sesiunii de shell (în cazul unei autentificări reușite), este prezentată în figura 6.8:

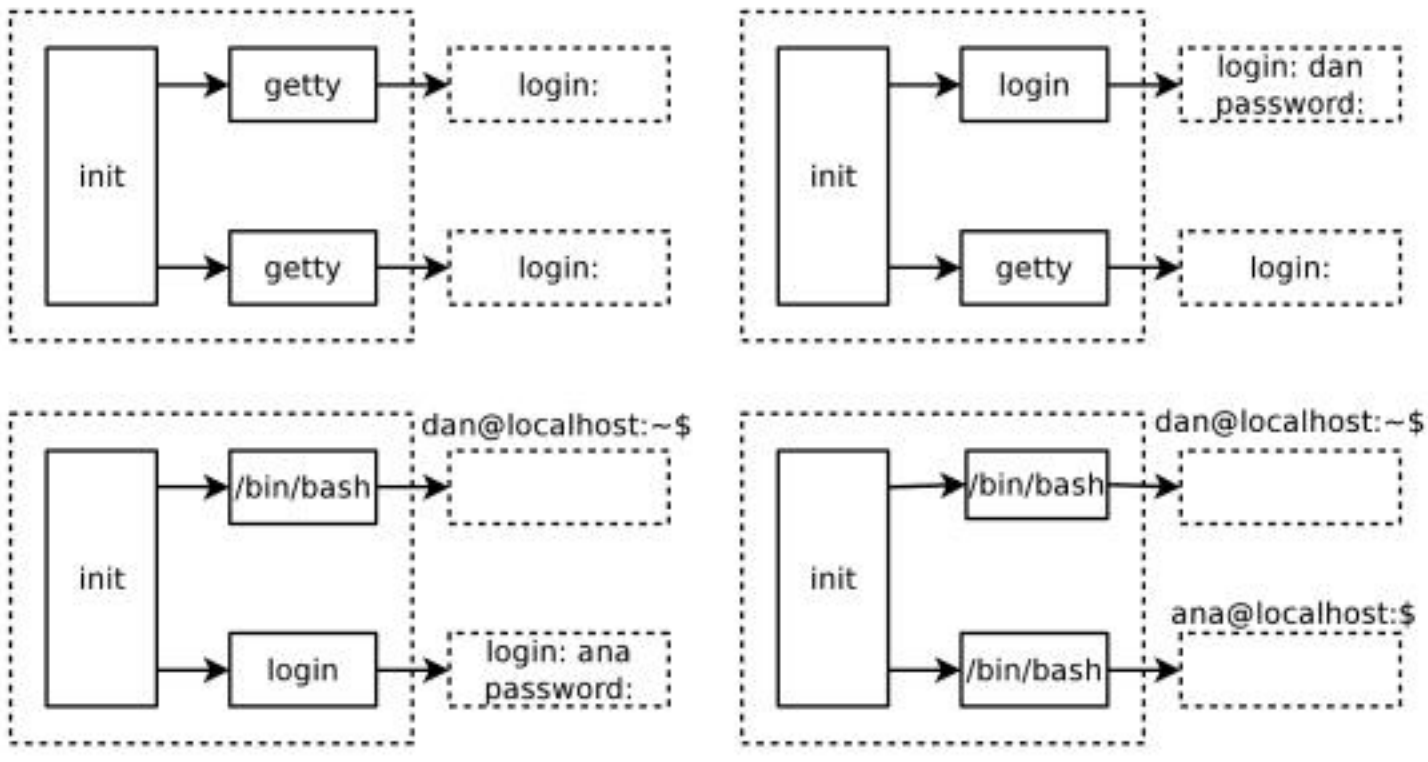


Figura 6.8: Autentificarea într-un sistem Linux

Se observă că se rulează getty pe toate terminalele. În momentul în care utilizatorul

introduce numele de utilizator, getty crează procesul login care cere parola utilizatorului. O dată cu introducerea și validarea parolei, utilizatorului îi este permis accesul în sistem și se deschide, în mod tipic, o sesiune shell. După cum se observă în figura 6.8, utilizatorul `dan` rulează shell-ul Bash (`/bin/bash`).

La încheierea sesiunii de shell curente, procesul getty este reponit așa cum este configurat în `/etc/inittab` (*respawn*).

Fișierele /etc/passwd și /etc/shadow

Fișierele `/etc/passwd` și `/etc/shadow` sunt utilizate pentru autentificarea utilizatorilor în sistem și stocarea de informații despre aceștia. După cum se poate observa dintr-o analiză simplă a fișierului `/etc/passwd`:

```
1 razvan@asgard:~$ cat /etc/passwd | grep guest
2 guest:x:1001:1001:Guest Account,,,:/home/guest:/bin/bash
```

O intrare din acest fișier este de forma:

```
1 username:simbol_parola:uid:gid:extra_info:home_dir:interpretor
```

Două dintre elementele prezentate sunt folosite de login pentru autentificarea utilizatorilor: `simbol_parola` și `interpretor`.

Câmpul `interpretor` specifică ce comandă va fi executată în momentul în care un utilizator a fost autentificat cu succes în sistem. De obicei această comandă este un interpretor de comenzi (în exemplul de mai sus `/bin/bash`), dar poate fi configurată orice altă comandă. În figura 6.8 utilizatorul `dan` folosește `/bin/bash`. Dacă se va completa intrarea asociată cu `/usr/bin/vim`, atunci orice autentificare pe un terminal virtual va rezulta în deschiderea editorului `vim` pe acel terminal.

Al doilea câmp dintr-o intrare în `/etc/passwd` (denumit aici `simbol_parola`) era folosit inițial pentru a stoca parola într-un format criptat. În felul acesta, oricine avea acces la fișier avea acces la parola criptată și putea folosi diverși algoritmi pentru a o sparge. O soluție ar fi fost restricționarea accesului la fișier. Totuși, acesta conține și alte informații necesare altor comenzi. Soluția aleasă a fost crearea unui nou fișier, `/etc/shadow`, care poate fi accesat doar de utilizatorul privilegiat.

```
1 root@anaconda:~# ls -l /etc/shadow
2 -rw-r----- 1 root shadow 1084 2009-07-21 22:19 /etc/shadow
```

Fișierul shadow conține numele de utilizator și parola criptată. Pentru mai multe detalii consultați paginile de manual (**`man 5 passwd`**, **`man 5 shadow`**) sau secțiunea 10.2.3.

6.5.4 Sesiune de shell

După rularea `init` și autentificarea utilizatorului în sistem, acestuia îi este oferită, în mod tipic, o sesiune de shell (conform configurației din fișierul `/etc/passwd`).

Înaintea oferirii promptului către utilizator, procesul shell nou creat citește un set de fișiere de configurare. Acestea sunt folosite pentru a stabili formatul promptului, umask-ul curent, aliasurile de comenzi, variabilele de mediu.

În cazul unui interpretor bash (*Bourne Again Shell*), fișierele de configurare citite sunt în ordine:

- `/etc/profile`: acesta este fișierul global de configurare pentru sesiunile de shell;
- `~/.bash_profile` sau `~/.bash_login` sau `~/.profile`: se execută primul găsit;
- `~/.bashrc` este fișierul obișnuit de configurare locală; drept urmare, `~/.bash_profile` conține linia

```
1 if [ -f ~/.bashrc ]; then . ~/.bashrc; fi
```

pentru a interpreta acest fișier.

După interpretarea fișierelor de configurare, se stabilește mediul curent de lucru și utilizatorului îi este oferit promptul unde poate introduce comenzi.

6.6 Studiu de caz

6.6.1 Grub 2

În ciuda faptului că nu a oferit nici o îmbunatățire majoră în ultimii ani, GRUB a rămas cel mai utilizat *bootloader* în cadrul distribuțiilor GNU/Linux. Aceasta datorită faptului că *bootloader*-ul este una din cele mai delicate componente ale sistemului, o actualizare greșită a sa ducând la nefuncționarea sistemului de operare.

Unul din elementele care au contribuit la lipsa de îmbunatățiri aduse GRUB este și arhitectura veche și greu de extins. În 2002 a început lucrul la o nouă versiune de GRUB (GRUB2), versiune care a adus aducă schimbări destul de mari.

GRUB2[1][2] a fost rescris de la zero pentru a asigura o portabilitate și o modularitate superioară versiunii anterioare. Printre avantajele pe care le oferă se numără:

- suport pentru scripturi;
- portabilitate pe mai multe arhitecturi;
- suport pentru i18n[3];
- un nou sistem de denumire a partițiilor.

Fișiere de configurare

GRUB2 se configurează într-un mod diferit față de GRUB. În primul rând numele fișierului de configurare s-a schimbat din `/boot/grub/menu.lst` în `/boot/grub/grub.cfg`. Spre deosebire de GRUB, noul fișier de configurare nu mai este editat direct de către utilizator, ci este generat automat:

[1] http://www.gnu.org/software/grub/grub-2.en.html
[2] https://wiki.ubuntu.com/Grub2
[3] http://en.wikipedia.org/wiki/Internationalization_and_localization

```
root@ant:~# cat /boot/grub/grub.cfg
#
# DO NOT EDIT THIS FILE
#
# It is automatically generated by /usr/sbin/grub-mkconfig using
templates
# from /etc/grub.d and settings from /etc/default/grub
#
[...]
```

Configurarea GRUB2 se realizează din trei locuri:

- fișierul `/etc/default/grub`: conține un set general de configurări, legate în principal de diverse opțiuni ale meniului de boot;
- directorul `/etc/grub.d`: conține un set de scripturi care descoperă imaginile de kernel instalate în sistem și generează `grub.cfg`;
- directorul `/boot/grub`: conține *bootloader*-ul și diversele module pe care acesta le folosește.

După editarea unuia din fișierele din `/etc/grub.d` sau a `/etc/default/grub` va trebui rulată comanda **`update-grub2`** pentru ca fișierul `/boot/grub/grub.cfg` să fie regenerat.

Editarea opțiunilor meniului

Fișierul `/etc/default/grub` conține configurări ale meniului de boot. Spre exemplu `GRUB_DEFAULT` specifică opțiunea din meniu selectată în mod implicit, iar `GRUB_TIMEOUT` specifică *timeout*-ul după care opțiunea implicită va fi utilizată pentru bootare.

```
root@ant:~# cat /etc/default/grub
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.

GRUB_DEFAULT=0
GRUB_TIMEOUT="10"
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""
[...]
```

Adăugarea unei noi opțiuni în meniu de boot

Opțiunile din meniul GRUB2 sunt create folosind scripturile din directorul `/etc/grub.d`:

- `00_header`: încarcă informațiile din `/etc/default/grub`;
- `05_debian_theme`: configurează fundaluri, culori și teme;
- `10_linux`: caută imagini de kernel pe disc și le introduce în meniu;

- `20_memtest86+`: introduce în meniu imaginea `/boot/memtest86+.bin`, dacă aceasta există;
- `30_os-prober`: caută alte imagini de sisteme de operare și le include în meniu;
- `40_custom`: permite utilizatorului să introducă intrări statice în meniu.

Ordinea în care apar opțiunile în meniu este dată de ordinea numelor fișierelor. Astfel, întotdeauna `memtest86+` (`20_memtest86+`) va apărea după imaginile de kernel de pe disc (`10_linux`).

Pentru a adăuga o nouă opțiune există două posibilități: se editează fișierul `40_custom` sau se poate crea un fișier nou, caz în care numele fișierului nou creat va defini poziția intrării în meniu.

Un exemplu de sintaxă care se folosește pentru adăugarea de noi opțiuni este:

```
1  echo "Adding Custom Kernel \& SystemRescue" >\&2
2  cat << EOF
3  menuentry "Ubuntu, linux 2.6.31-11-custom" {
4          set root=(hd0,9)
5          linux /boot/vmlinuz-2.6.31-11-custom root=UUID=c6829e27-2350-4e84
   -bdbb-91b83f018f98 ro
6          initrd /boot/initrd.img-2.6.28-11-generic
7  }
8
9  menuentry "Boot SystemRescue CD from hard drive" {
10         set root=(hd1,10)
11         linux   /sysrcd/rescuecd subdir=sysrcd setkmap=us
12         initrd  /sysrcd/initram.igz
13 }
14 EOF
```

Linia `echo "Adding Custom Kernel & SystemRescue" >&2` nu este neapărat necesară. Dacă este inclusă, atunci textul *Adding...* va fi afișat în timpul execuției comenzii **`update-grub2`**. Celelalte opțiuni sunt similare cu cele din GRUB.

Unul din elementele de noutate pe care le-a introdus GRUB2 este o nouă schema de denumire a partițiilor. Astfel, discurile sunt denumite în forma `hdX`, unde X este un numar ce începe de la 0 (spre exemplu, primul disc SATA, este denumit `hd0`, al doilea disc SDB este `hd1` etc.), similar cu GRUB. În schimb, partițiile se denumesc de forma `sdaY`, cu Y începând de la 1 (de exemplu a treia partiție de pe sda este `sda3`).

6.6.2 Pornirea sistemului de operare Windows XP

Pornirea sistemului de operare Windows XP urmărește aceeași pași ca în cazul pornirii Linux: pornirea sistemului, încărcarea BIOS, încărcarea *bootloader*-ului, încărcarea nucleului, pornirea serviciilor.

Pașii până la încărcarea bootloader-ului sunt independenți de sistemul de operare și, deci, identici cu cei pentru Linux.

Bootloader-ul pentru Windows NT (NT, 2000, XP, 2003) este **NT Loader** (NTLDR).

NTLoader

La fel ca și în cazul Linux, sectorul de boot al hard-disk-ului (MBR) sau al unei partiții (VBR) este limitat și este folosit pentru încărcarea NTLDR. Imaginea acestuia rezidă în `C:\Windows\ntldr`.

După încărcarea bootloader-ului, acesta consultă fișierul `boot.ini` pentru a localiza informațiile de pe partiția de sistem.

Configurare NTLoader (boot.ini)

Fișierul `boot.ini` se află pe partiția de sistem (`C:`). Pentru vizualizarea și accesarea acestuia trebuie permisă vizualizarea fișierelor de sistem (`Tools->Folder Options ->View->Hide protected operating system files`).

Un exemplu de conținut al `boot.ini` este prezentat în continuare:

```
1 [boot loader]
2 timeout=30
3 default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
4 [operating systems]
5 multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP
  Professional" /noexecute=optin /fastdetect
```

În cazul existenței mai multor sisteme Windows, utilizatorului îi va fi afișat un ecran de selecție (asemănător cu cel al GRUB).

Variabila `timeout` specifică timpul de așteptare înainte de a încărca sistemul de operare dat de opțiunea implicită.

Lista cu opțiunile posibile de boot-are a nucleului Windows XP se poate consulta aici[1].

Încărcarea nucleului

După rularea *bootloader*-ului, acesta încarcă în memorie imaginea de nucleu a Windows, `ntoskrnl.exe`, și `hall.dll` (*Hardware Abstraction Layer*), controlul fiind apoi oferit nuclelului.

Nucleul este încărcat în memorie în două faze: în prima fază se inițializează dispozitivele sistemului (procesor, memorie), iar în a doua fază se încarcă device driver-ele de sistem necesare.

Pornirea proceselor importante

După încărcarea nucleului în memorie, sunt pornite, în ordine, două procese importante:

- **smss** (*Session Manager Subsystem*): acesta montează dispozitivele din sistem, creează variabilele de mediu și pornește winlogon (*Windows Logon Manager*);

[1] http://support.microsoft.com/default.aspx?scid=kb;en-us;833721

- **winlogon** (*Windows Logon Manager*): este procesul care se ocupă de autentificarea utilizatorilor în sistem; în urma unei autentificări reușite sunt pornite shell-ul Windows (`explorer`) și alte procese ale utilizatorului.

Mai multe detalii despre procesul de pornire a unui sistem Windows XP se pot găsi aici[1].

6.6.3 Interoperabilitate Linux-Windows

În cazul sistemelor dual-boot Linux-Windows se poate întampla să se suprascrie MBR-ul și să se piardă astfel opțiunea directă de boot-are a unuia dintre sistemele de operare. Pentru a recupera *bootloader*-ul pierdut, trebuie rescris MBR-ul.

Reinstalarea GRUB

Situația cea mai frecventă ce necesită o reinstalare GRUB este aceea în care se reinstalează Windows pe un sistem pe care există instalată și o distribuție Linux. La instalarea Windows suprascrie MBR astfel încât se pierde *stage1* din GRUB.

Pentru a reinstala GRUB, este necesar un mediu boot-abil Linux. Acesta poate fi un LiveCD sau o dischetă boot-abilă. În cazul unei dischete boot-abile utilizatorului îi este oferit direct prompt-ul de grub. În cazul unui LiveCD, utilizatorul va trebui să deschidă o consolă și să acceseze de acolo consola de GRUB:

```
root@asgard:~# grub
Probing devices to guess BIOS drives. This may take a long time.

    GNU GRUB  version 0.97  (640K lower / 3072K upper memory)

       [ Minimal BASH-like line editing is supported.  For
         the  first  word,  TAB  lists  possible  command
         completions.  Anywhere else TAB lists the possible
         completions of a device/filename. ]

grub>
```

Presupunând că se dorește instalarea GRUB pe primul hard-disk din sistem, vor trebui urmați pașii de mai jos:

- specificarea partiției care conține imaginea `stage2` de GRUB:

```
grub> root (hd0,2)
```

- dacă nu se știe pe ce partiție se află imaginile se poate folosi comanda **`find`**, căutând unul din fișierele din `/boot`. Cel mai adesea este căutat `stage1`:

```
grub> find /boot/grub/stage1
 (hd0,2)
```

- după descoperirea partiției se realizează pasul 1, apoi se instalează imaginea de GRUB în MBR:

```
grub> setup (hd0)
```

[1] http://technet.microsoft.com/en-us/library/bb457123.aspx

- imaginea de GRUB poate fi instalată și în sectorul de boot al unei partiții (VBR):

```
grub> setup (hd0,0)
```

  însă, în acest caz, GRUB va trebui să fie pornit (*chainloaded*) cu un alt bootloader.

Comanda **`quit`** se folosește pentru ieșirea din consola GRUB. După repornirea sistemului, acesta va afișa ecranul de opțiuni GRUB pentru selecția sistemului de operare dorit.

Reinstalarea NTLoader

Reinstalarea NTLDR este necesară în cazul în care se instalează o distribuție Linux fără a se instala și un bootloader (spre exemplu GRUB), sau în cazul unui MBR corupt.

Pentru repararea MBR, trebuie folosită consola de recuperare Windows XP (*Recovery Console*). Pentru accesarea acesteia se folosește CD-ul de instalare Windows și după bootare se alege opțiunea *Repair*. În continuare se apasă `R` pentru rularea *Emergency Repair Process* și se alege *Fast* sau *Manual Repair*. *Fast Repair* este recomandată majorității utilizatorilor, iar *Manual Repair* administratorilor și utilizatorilor avansați.

Mai multe detalii găsiți online[1].

6.6.4 Personalizarea ecranului GRUB

Ecranul de opțiuni prezentat de GRUB poate fi anost. Pentru a rezolva acest neajuns se poate folosi o altă imagine de fundal, denumită GRUB *splashimage*. Există numeroase site-uri care oferă astfel de imagini[2][3][4]). Imaginile sunt imagini .xpm arhivate (de exemplu hubble.xpm.gz).

Pentru folosirea unei imagini splash de GRUB trebuie editat fișierul principal de configurare a GRUB (`/boot/grub/menu.lst`). Pașii ce trebuiesc urmați sunt prezentați mai jos:

- crearea unei copii de siguranță a fișierului de configurare GRUB:

```
root@asgard:/boot/grub# cp menu.lst menu.lst.old
```

- obținerea unui fișier .xpm.gz care conține imaginea de splash folosită pentru GRUB:

```
root@asgard:/boot/grub# wget http://schragehome.de/splash/hubble.xpm
.gz
--2009-09-26 06:23:27--  http://schragehome.de/splash/hubble.xpm.gz
Resolving schragehome.de... 82.165.105.215
Connecting to schragehome.de|82.165.105.215|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 99593 (97K) [image/x-xpixmap]
Saving to: 'hubble.xpm.gz'
```

[1] http://www.ntfs.com/missing-corrupted-system-files.htm
[2] http://schragehome.de/splash/
[3] http://www.queervisions.com/arch/2007/04/9_grub_splash_i.html
[4] http://kde-look.org/index.php?xcontentmode=35x45

```
8
9  100%[=====================================>] 99,593       209K/s   in
   0.5s
10
11 2009-09-26 06:23:27 (209 KB/s) - 'hubble.xpm.gz' saved [99593/99593]
```

- editarea fișierului de configurare:

```
1 [...]
2 # Pretty colours
3 #color cyan/blue white/blue
4 splashimage=(hd0,2)/boot/grub/hubble.xpm.gz
5 [...]
```

  în exemplul de mai sus partiția unde se află imaginea de GRUB este a treia partiție de pe primul disc (`hd0,2`).

După editarea și salvarea fișierului se repornește sistemul. Noul ecran de opțiuni GRUB va folosi ca imagine de fundal `hubble.xpm.gz`.

Cuvinte cheie

- pornirea/repornirea unui sistem
- booting/bootstrapping
- fazele pornirii unui sistem de operare
- BIOS
- POST
- CMOS
- dispozitiv boot-abil
- sector de boot/sector bootabil
- MBR, VBR
- tabelă de partiții
- bootloader
- fazele unui bootloader (stages)
- GRUB
- stage1, *_stage1_5, stage2
- /boot/grub/menu.lst
- consola GRUB
- imagine de nucleu
- opțiuni de încărcare a nucleului
- init
- runlevel
- /etc/inittab
- /etc/init.d, /etc/rcX.d
- getty
- login
- /etc/passwd, /etc/shadow
- /etc/profile, ~/.bash_profile, ~/.bashrc
- NTLDR
- boot.ini
- smss, winlogon
- Recovery Console

Întrebări

1. Care este ordinea corectă a pornirii unui sistem?
   - ❑ vmlinuz, init, GRUB, BIOS
   - ❑ GRUB, BIOS, init, vmlinuz

- ❑ BIOS, GRUB, vmlinuz, init
- ❑ init, vmlinuz, BIOS, GRUB

2. Ce este un sector de boot?
   - ❑ primul sector al primei partiții
   - ❑ primul sector al unei partiții sau al unui hard-disk
   - ❑ un sector ce conține semnătura 0xAA55 (ultimii doi octeți)
   - ❑ un sector pe care se găsește faza a doua a GRUB (stage2)
3. Care din următoarele este un înlocuitor al init?
   - ❑ upstart
   - ❑ downstart
   - ❑ slowstart
   - ❑ faststart
4. Câte partiții primare, active, extinse, logice se pot afla pe un hard-disk?
   - ❑ oricâte, 1, 2, 4
   - ❑ 1, oricâte, oricâte, 4
   - ❑ 4, 1, 1, oricâte
   - ❑ 1, oricâte, 2, oricâte
5. Care din următoarele asocieri NU este validă?
   - ❑ GRUB/NTLDR
   - ❑ vmlinuz/ntoskrnl
   - ❑ POST/CMOS
   - ❑ getty/winlogon
6. Care fază a GRUB este resposabilă cu încărcarea imaginii de nucleu Linux?
   - ❑ stage1
   - ❑ stage1.5
   - ❑ stage2
   - ❑ stage3
7. Care este fișierul principal de configurare GRUB pe un sistem Debian/Ubuntu?
   - ❑ /etc/main.cf
   - ❑ /etc/grub.conf
   - ❑ /boot/grub.conf
   - ❑ /boot/grub/menu.lst
8. Care este ordinea corectă de pornire a unei sesiuni shell?

- ❑ login, init, getty, bash
- ❑ getty,bash, login, init
- ❑ init, getty, login, bash
- ❑ init, login, bash, getty

9. Cum se reprezintă, în notația GRUB, a 3-a partiție de pe al doilea disc?
   - ❑ /dev/hdb3
   - ❑ /dev/hda2
   - ❑ (hd1,2)
   - ❑ (hd2,3)
10. Fișierul `/etc/shadow` conține parolele criptate ale utilizatorilor sistemului. Poate fi accesat numai de utilizatori privilegiați ai sistemului.
    - ❑ adevărat, adevărat
    - ❑ adevărat, fals
    - ❑ fals, adevărat
    - ❑ fals, fals

Capitolul 7

Analiza hardware a sistemului

It's hardware that makes a machine fast. It's software that makes a fast machine slow.

Craig Bruce

Ce se învață din acest capitol?

- Principalele componente hardware ale unui sistem de calcul
- Mecanismele de accesare și interogare a dispozitivelor hardware
- Încărcarea si folosirea modulelor de kernel
- Utilitare de interacțiune cu dispozitivele: **`lspci`**, **`lsusb`**, **`smartmontools`**
- Lucrul cu dispozitivele sistemului; comanda **`dd`**
- Interacțiunea cu sistemul: `/proc`, **`sysctl`**
- Cum se poate realiza o copie de siguranță pentru MBR

7.1 Structura unui sistem de operare

Sistemul de operare este cel care se ocupă de managementul resurselor hardware și software prezente în computer. SO are mai multe componente, fiecare cu un rol bine definit. Diagrama 7.1 prezintă modul în care este structurat accesul la hardware.

Sistemul de operare este compus din:

- **kernel**: nucleul sistemului de operare – se ocupă de comunicația directă cu hardware-ul și oferă un sistem de management al resurselor pentru aplicațiile din sistem;
- **biblioteci de funcții** – pun la dispoziție aplicațiilor un API (*Application Programming Interface*) comun pentru accesul la resursele oferite de kernel;
- **aplicații** – cele care vin cu un sistem de operare sunt de obicei utilizate pentru configurarea sistemului, modificarea datelor din sistem etc. Aceste aplicații pot să utilizeze biblioteci de funcții sau pot comunica direct cu kernel-ul.

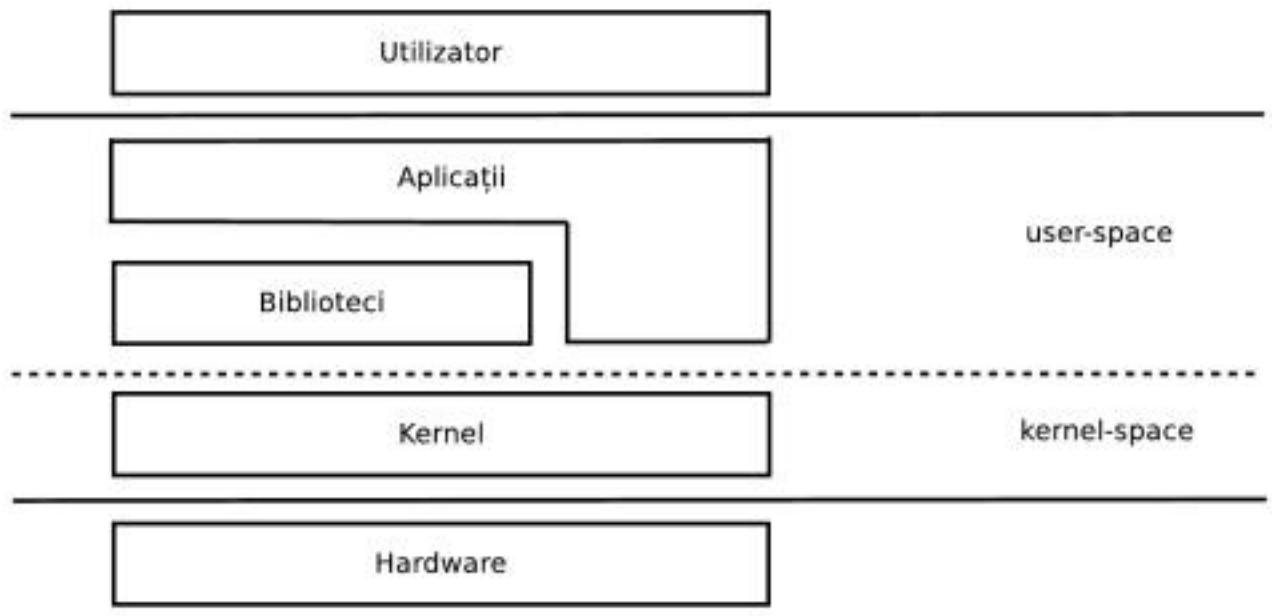


Figura 7.1: Accesul la hardware

Sistemul de operare face distincția între două "spații" în care se pot găsi programe în execuție: spațiul utilizator (*user-space*) și spațiul kernel (*kernel-space*). Prin "spațiu" se înțelege atât spațiu de memorie cât și mediu de execuție. Fiecare spațiu are alte condiții de lucru.

Spațiul kernel este spațiul în care rulează kernel-ul. Conține doar kernel-ul și componentele sale (precum driver-ele încărcate la un moment dat). Tot ceea ce se găsește în spațiul kernel are acces direct la hardware.

Spațiul utilizator este spațiul în care se găsesc aplicațiile și bibliotecile de funcții utilizate de acestea la un moment dat. Tot ceea ce se găsește în spațiul utilizator nu are acces direct la hardware.

Toate acțiunile aplicațiilor din spațiul utilizator trebuie să treacă prin kernel. Comunicația dintre spațiul utilizator și spațiul kernel se realizează prin apeluri de sistem.

Apelul de sistem este un mecanism prin care programele sau bibliotecile din spațiul utilizator accesează resursele puse la dispoziție de kernel.

Practic, apelurile de sistem sunt funcții implementate în kernel pe care aplicațiile din spațiul utilizator le accesează direct sau prin intermediul bibliotecilor de funcții.

Acest capitol va analiza modul de lucru intern pentru un sistem Linux.

7.2 Considerente hardware

Componentele fizice ale sistemului poartă denumirea generică de **hardware**. Pentru folosirea acestora, utilizatorul dispune de un sistem de operare și aplicații. Acestea reprezintă nivelul de software de peste hardware. Fiecare componentă este esențială și nu poate funcționa în lipsa celeilalte.

Sistemul de operare și aplicațiile de bază sunt cele care asigură interfața cu hardware-ul pentru a preveni, astfel, dificultatea programării directe a hardware-ului. Din punctul de

vedere al utilizatorului, interfața cu hardware-ul este limitată la un set de apeluri către sistemul de operare și aplicațiile de bază.

7.2.1 Structura unui sistem de calcul

Din punct de vedere structural, un sistem de calcul este un set de componente cu roluri bine definite care comunică pentru îndeplinirea unei sarcini. Sistemele de calcul din ziua de azi urmează structura von Neumann prezentată mai jos. Aceasta este folosită pentru a defini setul de componente care există într-un sistem de calcul și interacțiunea dintre acestea.

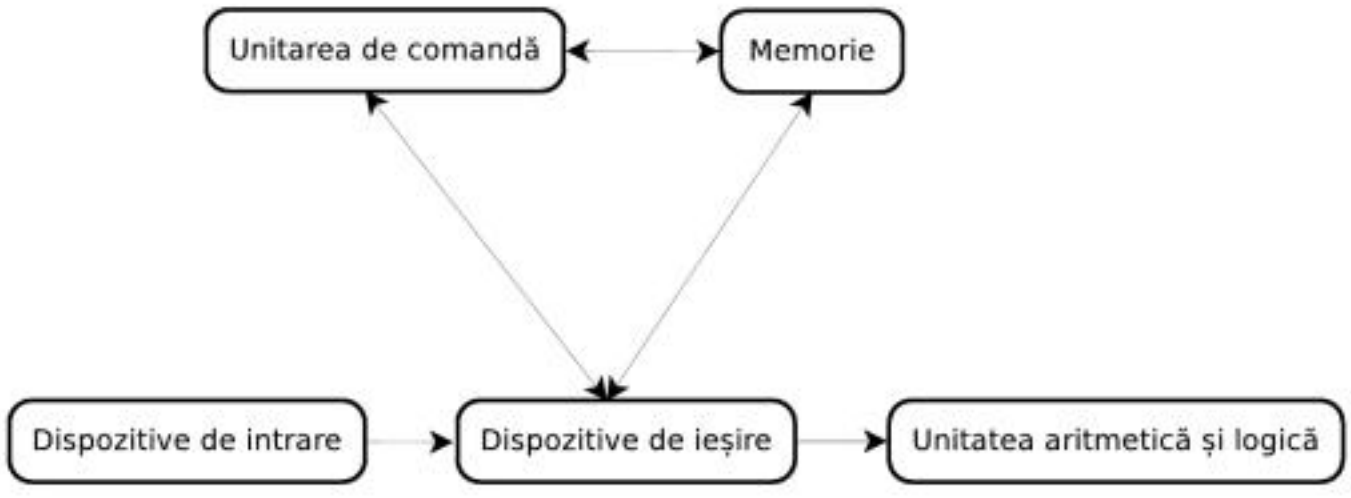


Figura 7.2: Interacțiunea componentelor hardware ale unui sistem de calcul

După cum se observă, arhitectura von Neumann precizează 5 componente ale unui sistem de calcul:

- **unitatea de comandă** sau **unitatea de control** este responsabilă cu comandarea modului în care celelalte componente lucrează; unitatea de comandă va interpreta codul unui program (va decodifica instrucțiunile) și va cere unității aritmetice și logice execuția sarcinilor necesare;

- **unitatea aritmetică și logică** se mai numește **unitatea de execuție**; rolul acesteia este de a executa codul transmis de la unitatea de comandă; unitatea aritmetică are un set de registre pentru stocarea locală a informației și un acumulator pentru efectuarea de operații; rolul unității aritmetice și logice este îndeplinit de procesor;

- **memoria** este componenta în care se stochează codul programului de executat și datele acestuia; memoria este folosită de unitatea de execuție și de comandă; unitatea de comandă citește din memorie instrucțiunile și le decodifică, pe când unitatea de execuție citește și scrie în memorie datele cu care efectuează diverse calcule;

- **dispozitivele de intrare/ieșire (periferice)** sunt cele care intermediază comunicația sistemului cu exteriorul; informațiile de la aceste dispozitive sunt copiate în memorie și folosite ulterior de unitatea de comandă și unitatea de execuție; exemple de dispozitive periferice sunt monitorul, tastatura, mouse-ul, hard disk-ul etc.

7.2.2 Procesorul

Procesorul sau unitatea centrală de prelucrare (CPU – *Central Processing Unit*) este "creierul" sistemului. Este echivalentul unității de comandă și a celei de execuție din arhitectura von Neumann. Procesorul este cel care va interpreta codul unui program și va efectua operații cu datele acestuia și este componenta de bază a execuției unui proces.

Procesorul este cel care determină arhitectura unui sistem de calcul. Astfel, procesoarele Intel sau AMD fac parte din arhitectura `x86`. Majoritatea procesoarelor mai noi fac parte din arhitectura `x86_64`. Arhitectura unui sistem de calcul se referă la setul de instrucțiuni pe care procesorul le oferă utilizatorului.

Procesorul dispune de un set de locații de stocare de foarte mare viteză denumite **registre**. Viteza procesorului este foarte mare comparativ cu viteza memoriei și de aceea este nevoie de registre pentru a efectua calcule rapide. În ziua de azi dimensiunea registrelor este de 32 sau de 64 de biți. Dimensiunea registrelor este sursa denumirii sistemului. Un sistem este denumit sistem pe 32 de biți sau sistem pe 64 de biți în funcție de dimensiunea registrelor sale.

Un procesor poate avea una sau mai multe unități de procesare (*core*). O unitate de procesare este un circuit electronic care execută instrucțiuni. Un procesor cu o singură unitate de procesare se numește **single-core**, iar un procesor cu două sau mai multe unități de procesare aflate pe același circuit integrat se numește **multi-core**. Performanța obținută prin folosirea mai multor unitati de procesare depinde de existența firelor de execuție multiple în cadrul aplicațiilor și de suportul sistemului de operare.

Caracteristica vitezei unui procesor este, de obicei, **frecvența** acestuia, măsurată în Hz. În zilele noastre, frecvența de lucru a unui procesor este de ordinul GHz. Calculatoarele persoanele folosesc procesoare cu arhitectură i386, precum: Intel Pentium, Intel Xeon, AMD Athlon, AMD Opteron etc.

Există un număr însemnat de arhitecturi menite să adreseze de la constrângeri specifice: spre exemplu cele pentru platforme integrate (embedded systems): ARM, MIPS, sau alte platforme de calcul intensiv: Alpha, PowerPC etc.

Pentru arhitecturi diferite poate varia numărul de biti pe care sunt efectuate operațiile sau numărul de registre aflate la dispoziția procesorului. Astfel nu putem compara viteza unui procesor i386 cu un procesor PowerPC folosind frecvența.

7.2.3 Ierarhia de memorie

În arhitectura unui calculator modern, memoria este organizată în mod ierarhic. Fiecare nivel din ierarhie are o lățime de bandă mai mare, dimensiune mai mică și latență mai scăzută decât nivelul inferior. După cum se observă în figura 7.3 cele 4 niveluri principale sunt: registrele procesorului, memoria cache, memoria RAM și hard disk-ul.

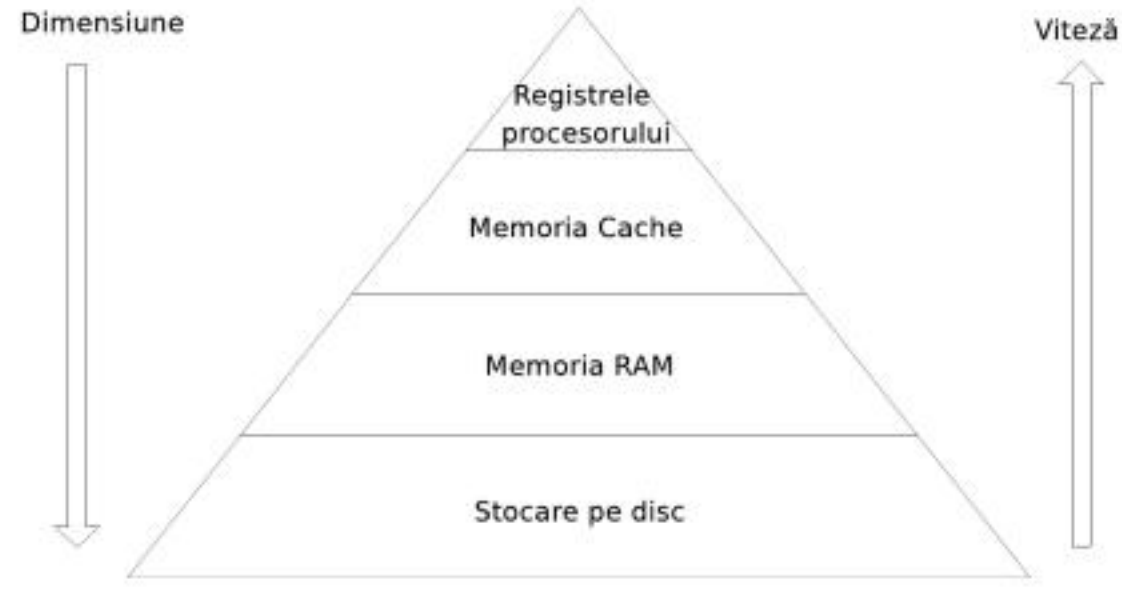


Figura 7.3: Ierarhia de memorie

Registrele procesorului

În vârful ierarhiei de memorii se află registrele procesorului care au cea mai mare viteză de acces, de obicei un ciclu de CPU, dar dimensiunea de câteva sute de bytes. Ele sunt folosite pentru a stoca datele care sunt prelucrate de către procesor la momentul curent.

Memoria cache

Din cauza diferenței mari de viteză între memoria RAM și procesor, se folosește un nivel intermediar de memorie: mai rapidă dar mai scumpă și de dimensiunea mai mică. Această memorie poartă numele de **memorie cache** și este folosită pentru reținerea anumitor informații din memoria RAM.

Memoria cache folosește conceptul de localizare temporală și spațială. Aceasta înseamnă că în momentul în care procesorul dorește să acceseze o zonă de memorie, mai întâi va compara adresa acesteia cu adresele zonelor de memorie care au fost copiate deja în cache până în acel moment. Dacă descoperă că această zonă există în cache, ea va fi accesată direct de acolo. În caz contrar, zona de memorie dorită va fi copiată în cache pentru a putea fi refolosită în momentele apropiate. Când cache-ul se umple, intrările din el sunt șterse în ordinea vechimii lor. În mod tipic, memoria cache are o dimensiune de ordinul sutelor de KB sau câtorva MB.

Există trei tipuri de memorie cache: L1, L2 și L3.

Memoria cache L1, sau de nivel 1, este o memorie încorporată în procesor, de dimensiune foarte mică dar foarte rapidă, fiind folosită pentru a accesa date importante și frecvent folosite de către procesor. Poate fi accesată în câtiva cicli de CPU și are dimensiunea de zeci de KB. Memoria cache L1 este de obicei de tipul SRAM (Static RAM), ceea ce înseamnă că nu necesită refresh pentru a menține datele.

Memoria cache L2 mai este numită și cache secundar, fiind folosită pentru a stoca date folosite recent. De obicei este externă procesorului dar se tinde spre încorporarea ei în

procesor. Poate fi folosită ca buffer pentru datele și instrucțiunile ce urmează a fi preluate din memoria RAM. Are o latență de 2 până la 10 ori mai mare decât memoria cache L1, dar dimensiunea mult mai mare. Inițial dimensiunea memoriei cache L1 era de 256 sau de 512 KB, iar în prezent are dimensiunea de ordinul MB.

Memoria cache L3 este o memorie specializată care cooperează cu memoria cache L1 și L2 pentru a îmbunătăți performanțele sistemului. Are dimensiunea mai mare decât memoriile L1 și L2, fiind de ordinul MB. Deși latența introdusă este mai mare decât pentru L1 și L2, accesarea ei este mai rapidă decât în cazul memoriei RAM.

Memoria RAM

Când ne referim la memoria unui sistem de calcul ne vom referi, de obicei, la memoria RAM (*Random Access Memory*). Aceasta este folosită pentru a stoca datele și codul unui proces. În momentul în care se dorește efectuarea unui calcul sau decodificarea unei instrucțiuni, se face un apel la memorie pentru citirea acelei informații.

În zilele noastre, memoria RAM folosită este de tipul DRAM (Dynamic RAM). Tipul cel mai recent este DDR3. Caracteristicile importante ale unui modul de memorie RAM sunt:

- frecvența de lucru: de ordinul MHz
- timpul de acces: de ordinul nanosecundelor
- rata de transfer: măsurată în GB/s
- capacitatea: actualmente de ordinul GB

Hard disk-ul

Probabil cel mai important dispozitiv periferic al sistemului este hard-disk-ul (HDD – *hard disk drive*). Acesta este folosit, de obicei, ca suport fizic pentru sistemele de fișiere. Hard-disk-urile reprezintă dispozitive de stocare permanentă (nepersistentă) a datelor.

Parametrii importanți care caracterizează un hard-disk sunt:

- capacitatea acestuia: în ziua de azi de ordinul sutelor de GB sau ordinul TB;
- viteza (măsurată în RPM – *Rotations Per Minute*): cu valori tipice de 5400 RPM, 7200 RPM, 10000 RPM;
- dimensiunea buffer-ului (un cache folosit pentru mărirea vitezei): cu valori de ordinul MB.

Există un set de interfețe și magistrale prin care un hard-disk poate fi accesat. Dintre acestea amintim ATA, Serial ATA, SCSI, SAS etc. Fiecare dintre aceste magistrale oferă o interfață distinctă de conectare a hard-disk-ului la placa de bază și un alt protocol de comunicare cu hard-disk-ul pentru accesarea informațiilor de pe acesta.

Un hard disk conține unul sau mai multe discuri neflexibile de metal numite platane. În cazul unui disc formatat, suprafața este împărțită în piste care sunt cercuri concentrice pe fiecare parte a platanelor. Pistele egal distantate de ax de pe fiecare parte a platanului

și de pe toate platanele, sunt grupate în cilindri. Fiecare pistă este împărțită în sectoare de câte 512 biți fiecare.

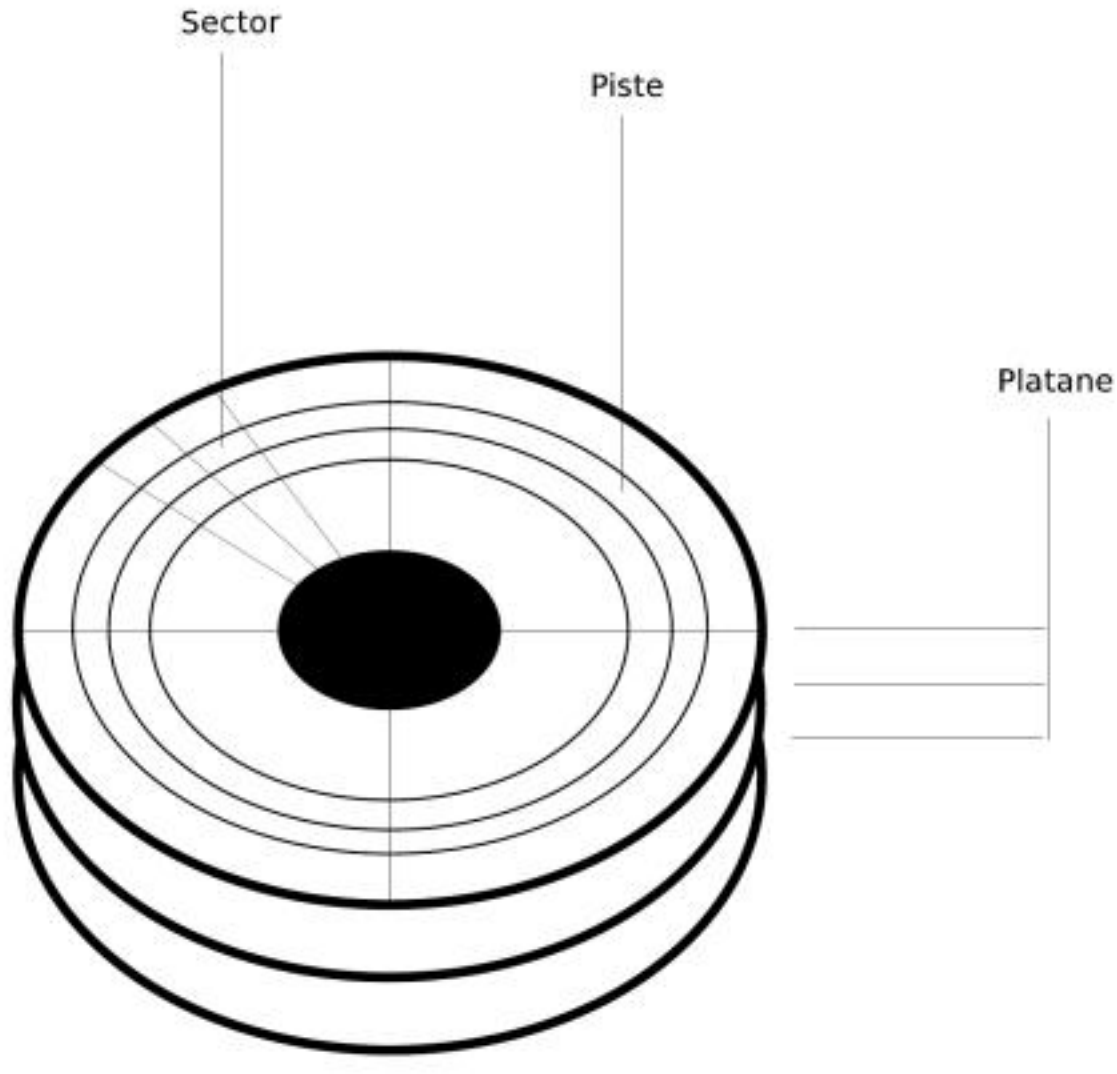


Figura 7.4: Structura unui hard-disk

7.2.4 Placa de bază; interconectarea componentelor

> **Placa de bază** (*mainboard*, *motherboard*) este componenta fizică al cărei rol este asigurarea conectivității celorlalte componente fizice. În mod tipic, o placă de bază are locuri (*sloturi*, *socket-uri*) pentru celelalte componente: procesor, memorie, placă video etc.

După cum se observă și în figura 7.5, placa de bază conține două "nuclee" de interconectare denumite **chipset-uri**. Acestea sunt *northbridge* și *southbridge*. Așa cum rezultă și din denumirea lor, *northbridge*-ul se află în partea de sus a plăcii de bază iar *southbridge*-ul în partea de jos. Există o conexiune a *northbridge* cu *southbridge*-ul.

Northbridge-ul mai este numit și *Memory Controller Hub (MCH)* și este folosit pentru a conecta componentele rapide ale unui sistem de calcul:

- Procesorul
- Memoria RAM
- Placa video (prin PCI Express sau AGP)

Northbridge-ul dictează tipul de procesor, cantitatea, viteza și tipul de memorie RAM care poate fi folosită.

Southbridge-ul mai este numit și *I/O Controller Hub (ICH)* și este folosit pentru interconectarea componentelor mai lente ale sistemului. Conține următoarele componente și magistrale:

- Magistrală PCI pentru legătura cu plăcile PCI
- Magistrală LPC pentru legătura cu Super I/O (tastatură, mouse, port paralel, port serial, controler de floppy)
- Magistrală SPI pentru legătura cu dispozitive de stocare de tip flash
- Magistrală SM pentru legătura cu alte dispozitive de pe placa de baza (senzori de temperatură și ventilator)
- Controler DMA
- Controler de întreruperi
- Dispozitivele de stocare (IDE, SATA)
- Real Time Clock (RTC)
- USB
- Ethernet (Interfața de rețea)
- Interfața audio
- Memorie BIOS

7.2.5 Dispozitive periferice; magistrale

Dispozitivele periferice ale sistemului asigură comunicația acestuia cu exteriorul. Conectarea acestor dispozitive periferice se realizează prin intermediul unor conectori în placa de bază. Acești conectori sunt oferiți fie direct de placa de bază (în cazul unor plăci incorporate), fie de o placă specializată conectată pe placa de bază (placa video, placa de rețea etc.)

Conexiunea diverselor plăci în placa de bază se realizează prin intermediul unor sloturi și a unor magistrale dedicate: PCI, AGP, USB etc.

Magistrale importante ale sistemului sunt:

- FSB – Front Side Bus: conectează procesorul de northbridge
- Back Side Bus – conectează memoria cache de procesor
- PCI – Peripheral Component Interconnection
- AGP – Advanced Graphics Port
- PCI Express
- USB – Universal Serial Bus

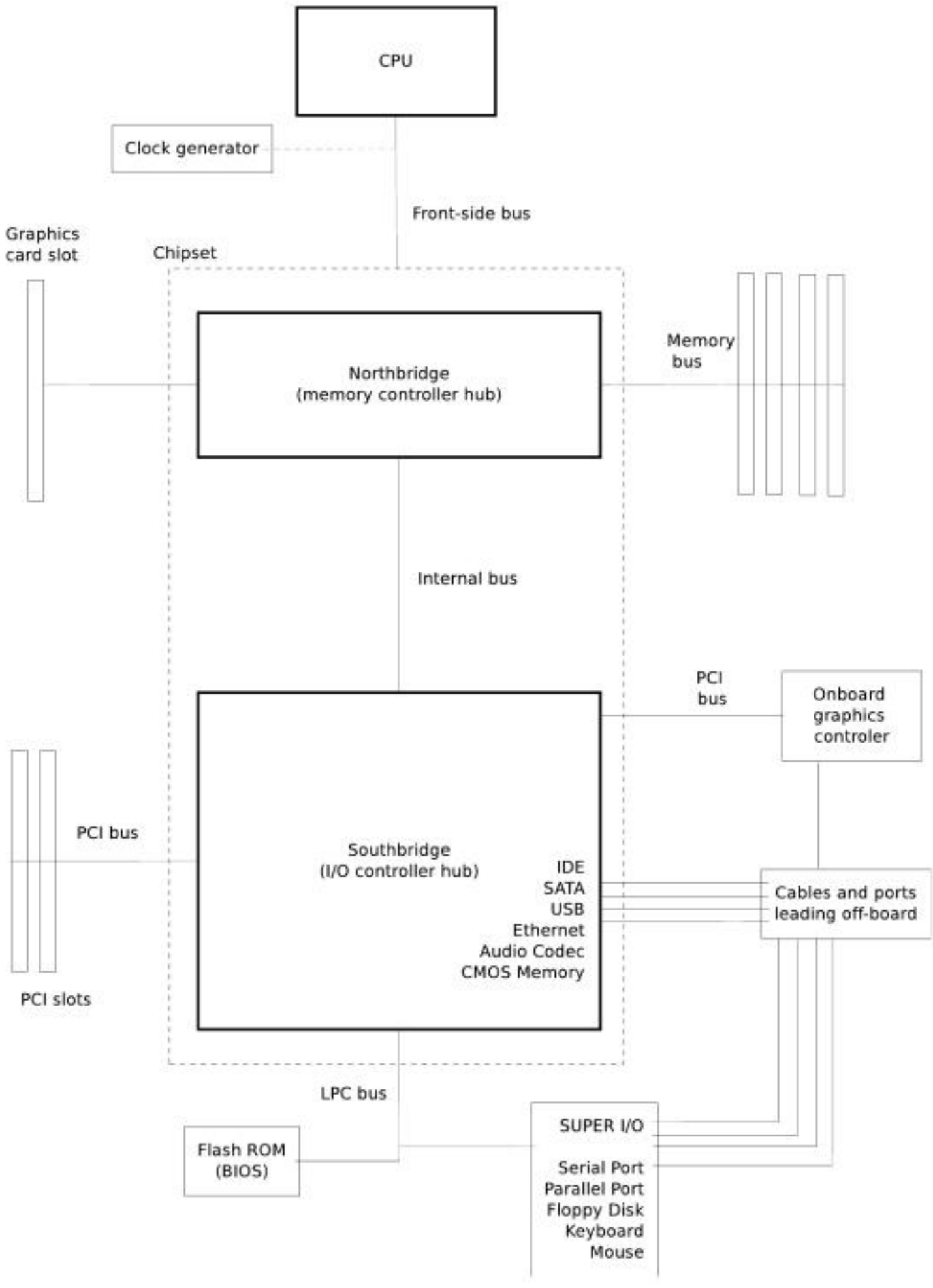


Figura 7.5: Diagrama bloc a plăcii de bază

7.3 Suportul pentru dispozitive la nivelul kernel-ului

Nucleul sistemului de operare este cel care intermediază accesul aplicațiilor la hardware. Pentru ca hardware-ul să poată fi utilizat, nucleul trebuie să știe cum să comunice cu acesta. Nucleul oferă suport de comunicație cu dispozitivele sistemului prin intermediul driver-elor.

Driver-ul reprezintă un set de rutine care se execută la nivelul nucleului și care permit accesul la un anumit dispozitiv.

Dispozitivele pe care le poate utiliza o aplicație pot fi dispozitive fizice reale sau pot fi dispozitive virtuale (care simulează comportamentul unui dispozitiv hardware). În Linux, suportul pentru dispozitive poate fi inclus în kernel (în momentul compilării) sau poate fi disponibil prin intermediul *modulelor*, module care pot fi încărcate la cerere de către sistem.

În prezent, majoritatea distribuțiilor conțin kernel-uri compilate cu suport generic pentru unele dispozitive mai des întâlnite, iar suportul pentru anumite dispozitive este disponibil prin module. Modulele sunt încărcate doar dacă dispozitivele aferente sunt prezente.

7.3.1 Listarea modulelor încărcate la un moment dat în sistem

Pentru a lista modulele încărcate la un moment dat se utilizează comanda **`lsmod`**:

```
ubuntu@ubuntu:~$ lsmod
Module                  Size  Used by
rfcomm                 40856  0
l2cap                  25728  5 rfcomm
bluetooth              55908  4 rfcomm,l2cap
lp                     12452  0
cpufreq_userspace       5408  0
cpufreq_powersave       2688  0
cpufreq_ondemand        9228  0
freq_table              5792  2 cpufreq_stats,cpufreq_ondemand
[...]
```

Comanda afișează pentru fiecare modul în parte, pe coloane:

- numele;
- spațiul de memorie ocupat de acel modul;
- numărul și numele modulele care depind de el.

Același lucru se poate afla prin intermediul sistemului de fișiere virtual `/proc`. Fișierul `/proc/modules` conține lista modulelor încărcate la un moment dat.

```
ubuntu@ubuntu:~$ cat /proc/modules
rfcomm 40856 0 - Live 0xe0bbb000
l2cap 25728 5 rfcomm, Live 0xe0b47000
bluetooth 55908 4 rfcomm,l2cap, Live 0xe0b51000
lp 12452 0 - Live 0xe0b39000
cpufreq_userspace 5408 0 - Live 0xe0b31000
cpufreq_stats 7360 0 - Live 0xe0b2e000
cpufreq_powersave 2688 0 - Live 0xe0859000
cpufreq_ondemand 9228 0 - Live 0xe0aee000
freq_table 5792 2 cpufreq_stats,cpufreq_ondemand, Live 0xe0b27000
[...]
```

Modulele disponibile în sistem se găsesc în `/lib/modules/<versiune_kernel>`. Modulele incluse în nucleul distribuției se găsesc în `/lib/modules/<versiune_kernel>/kernel`. În acest director ele sunt

organizate în funcție de categoria din care fac parte – există astfel directoarele `arch` (module pentru arhitectura curentă), `crypto` (criptografie), `drivers` (driver-e dispozitive), `fs` (sisteme de fișiere), `lib` (module care implementează diferite funcții în kernel), `net` (rețea), `security` (securitate), `sound` (sunet):

```
root@ubuntu:~# ls /lib/modules/2.6.18-4-686/kernel/
arch  crypto  drivers  fs  lib  net  sound
```

Versiunea de kernel poate fi aflată utilizând comanda:

```
ubuntu@ubuntu:~$ uname -r
2.6.20-15-generic
```

După cum s-a observat și din ieșirea comenzii **lsmod**, între module există dependențe. Aceste dependențe sunt descrise în `/lib/modules/<versiune_kernel>/modules.dep` – acest fișier este generat automat la rularea comenzii **depmod**.

7.3.2 Încărcarea unui modul

Înainte de a prezenta modul în care un modul este încărcat în nucleu, trebuie menționat că în distribuțiile moderne Linux modulele se încarcă automat în funcție de dispozitivele hardware prezente în sistem.

Încărcarea manuală a unui modul se poate realiza folosind comanda **insmod** sau comanda **modprobe**. Comanda **insmod** primește ca argument calea către fișierul unde se găsește modulul. Acest lucru o face destul de dificilă de folosit. Un exemplu de utilizare este dat mai jos (se presupune că modulul ce se dorește a fi încărcat nu este deja încărcat).

```
root@ubuntu:~# insmod ./drivers/input/misc/pcspkr.ko

root@ubuntu:~# lsmod | grep pcsp
pcspkr                  4224  0
```

O comandă echivalentă comenzii **insmod** utilizate mai sus dar care încarcă și dependențele unui modul (dacă există) este **modprobe <nume_modul>**:

```
root@ubuntu:~# modprobe pcspkr
```

Pentru a configura **modprobe** se poate utiliza fișierul `/etc/modprobe.conf` sau directorul `/etc/modprobe.d`.

7.3.3 Descărcarea unui modul din kernel

Pentru descărcarea unui modul din kernel se poate utiliza comanda **rmmod** sau comanda **modprobe -r**. Comenzile următoare descarcă modulul `pcspkr`, cu observația că, la utilizarea lui **modprobe**, se încearcă și descărcarea modulelor de care `pcspkr` depinde (dacă există și nu depind și alte module de ele).

```
root@ubuntu:~# rmmod pcspkr

root@ubuntu:~# modprobe -r pcspkr
```

Tabelul 7.1: Ierarhia din /sys

Director	Conținut
`/sys/devices`	Subdirectoarele care reflectă modul fizic de conectare al dispozitivelor în sistem
`/sys/bus`	Legături simbolice către dispozitive grupate în funcție de bus-ul (magistrala) de care aparțin acestea
`/sys/class`	Dispozitivele grupate în funcție de clasa din care fac parte
`/sys/block`	Dispozitivele bloc
`/sys/firmware`	Dispozitivele cu suport firmware
`/sys/fs`	Informații despre sistemele de fișiere
`/sys/kernel`	Informații despre kernel
`/sys/module`	Subdirectoare pentru fiecare modul în parte
`/sys/power`	Informații despre consumul de curent

7.4 Analiza hardware a unui sistem (magistrale, chipset, CPU, memorie, dispozitive)

7.4.1 Lista hardware – /sys

Folosind sistemul de fișiere virtual (`sysfs`) montat de kernel în `/sys`, se pot obține informații despre dispozitivele hardware aflate în system. `/sys` este populat de kernel cu informații provenite de la driver-ele încărcate la un moment dat. Dispozitivele hardware afișate în acest director sunt cele care au fost detectate de kernel și care sunt utilizabile de acesta. Tabelul 7.1 prezintă conținutul directoarelor din `/sys`.

7.4.2 Comenzi pentru afișarea dispozitivelor

Informații despre dispozitivele din sistem și starea acestora pot fi aflate și prin intermediul unor comenzi. În cele ce urmează, se vor prezenta comenzile utilizate pentru a prezenta diverse informații despre dispozitivele din sistem.

Dispozitive USB

Pentru a afișa magistralele și dispozitivele USB prezente în sistem se utilizează comanda **`lsusb`**:

```
1 ubuntu@ubuntu:~$ lsusb
2 Bus 002 Device 001: ID 0000:0000
3 Bus 001 Device 001: ID 0000:0000
```

Informații despre magistrala USB se regăsesc și în `/sys/bus/usb/`. Subdirectorul `devices` prezintă dispozitivele sub formă de legături simbolice către directoare din `/sys/devices`. Subdirectorul `drivers` din `/sys/bus/usb/` conține informații despre driver-ele utilizate.

```
ubuntu@ubuntu:~$ ls -l /sys/bus/usb/devices/
total 0
lrwxrwxrwx 1 root root 0 2007-09-04 18:16 1-0:1.0 ->
../../../devices/pci0000:00/0000:00:07.2/usb1/1-0:1.0
lrwxrwxrwx 1 root root 0 2007-09-04 18:16 2-0:1.0 ->
../../../devices/pci0000:00/0000:00:11.0/0000:02:02.0/usb2/2-0:1.0
lrwxrwxrwx 1 root root 0 2007-09-04 18:16 usb1 ->
../../../devices/pci0000:00/0000:00:07.2/usb1
lrwxrwxrwx 1 root root 0 2007-09-04 18:16 usb2 ->
../../../devices/pci0000:00/0000:00:11.0/0000:02:02.0/usb2

ubuntu@ubuntu:~$ ls -l /sys/bus/usb/drivers/
total 0
drwxr-xr-x 2 root root 0 2007-09-04 18:16 hub
drwxr-xr-x 2 root root 0 2007-09-04 18:16 usb
drwxr-xr-x 2 root root 0 2007-09-04 18:16 usbfs

ubuntu@ubuntu:~$ ls -lR /sys/bus/usb/drivers/ | grep module
lrwxrwxrwx 1 root root    0 module -> ../../../../module/usbcore
lrwxrwxrwx 1 root root    0 module -> ../../../../module/usbcore
lrwxrwxrwx 1 root root    0 module -> ../../../../module/usbcore
```

Ultima comandă listează recursiv directoarele din `/sys/bus/usb/drivers`, căutând liniile care conțin referințe la fișiere cu numele module. Se observă astfel că modulul utilizat este usbcore (datele și orele fișierelor au fost eliminate din ieșirea comenzii).

Dispozitive PCI

Dispozitivele de pe magistrala PCI se pot afla utilizând comanda **`lspci`**:

```
ubuntu@ubuntu:~$ lspci
00:00.0 Host bridge: Intel Corporation 82865G/PE/P DRAM Controller/Host-
Hub
Interface (rev 02)
00:01.0 PCI bridge: Intel Corporation 82865G/PE/P PCI to AGP Controller
(rev 02)
00:1d.0 USB Controller: Intel Corporation 82801EB/ER (ICH5/ICH5R) USB
UHCI
Controller #1 (rev 02)
00:1e.0 PCI bridge: Intel Corporation 82801 PCI Bridge (rev c2)
00:1f.0 ISA bridge: Intel Corporation 82801EB/ER (ICH5/ICH5R) LPC
Interface
Bridge (rev 02)
[...]
```

Similar magistralei USB, se pot determina și directoarele din `/sys` asociate dispozitivelor PCI. Comanda următoare prezintă modul în care se pot afla modulele utilizate de dispozitivele PCI, folosind aceeași procedură ca în cazul dispozitivelor USB (datele și orele fișierelor au fost eliminate din ieșirea comenzii).

```
ubuntu@ubuntu:~$ ls -lR /sys/bus/pci/drivers/ | grep module
lrwxrwxrwx 1 root root    0 module -> ../../../../module/snd_intel8x0
lrwxrwxrwx 1 root root    0 module -> ../../../../module/intel_agp
lrwxrwxrwx 1 root root    0 module -> ../../../../module/ata_generic
lrwxrwxrwx 1 root root    0 module -> ../../../../module/ata_piix
[...]
```

Sistemul ACPI

Sistemele moderne au implementat sistemul ACPI[1] (*Advanced Configuration and Power Interface*) pentru configurarea unor dispozitive și managementul consumului de curent. Fișierele virtuale ale acestui sistem se află în `/proc/acpi`.

Spre exemplu, sistemul ACPI oferă, printre altele, informații despre:

- procesoarele din sistem – `/proc/acpi/processor`
- ventilatoarele din sistem – `/proc/acpi/fan`
- bateriile din sistem – `/proc/acpi/battery`
- senzorii de temperatură – `/proc/acpi/thermal_zone`

hwinfo

`hwinfo` este o aplicație pentru detecția hardware-ului. Comanda **`hwinfo`** afișează în mod implicit toate informațiile despre dispozitivele detectate la un nivel foarte detaliat. Pentru o listare mai succintă se folosește următoarea comandă:

```
1  ubuntu@ubuntu:~$ hwinfo --short
2  cpu:
3                       Intel(R) Pentium(R) Dual  CPU  T2390  @ 1.86GHz,
   800
4  MHz
5                       Intel(R) Pentium(R) Dual  CPU  T2390  @ 1.86GHz,
6  1600 MHz
7  keyboard:
8    /dev/input/event4    AT Translated Set 2 keyboard
9  mouse:
10   /dev/input/mice      Macintosh mouse button emulation
11   /dev/input/mice      SynPS/2 Synaptics TouchPad
12 printer:
13   /dev/usb/lp0         Samsung ML-1640 Series
14 graphics card:
15                      Intel Mobile GM965/GL960 Integrated Graphics
16 Controller
17                      Intel 965 GM
18 sound:
19                      Intel 82801H (ICH8 Family) HD Audio Controller
20 storage:
21                      Intel 82801HBM/HEM (ICH8M/ICH8M-E) SATA IDE
22 Controller
23                      Intel 82801HBM/HEM (ICH8M/ICH8M-E) IDE Controller
24 network:
25   wlan0               Intel PRO/Wireless 3945ABG Network Connection
26   eth0                Realtek RTL8101E PCI Express Fast Ethernet
27 controller
28 network interface:
29   pan0                Ethernet network interface
30   wlan0               WLAN network interface
31   wmaster0            Network Interface
32   eth0                Ethernet network interface
```

[1] ACPI – http://www.acpi.info/

```
  lo                    Loopback network interface
disk:
  /dev/sda              WDC WD2500BEVS-2
[...]
```

Pentru a obține detalii despre o anumită componentă, se folosește numele componentei ca argument al comenzii. De exemplu, pentru procesor se folosește următoarea comandă:

```
ubuntu@ubuntu:~$ hwinfo --cpu
01: None 00.0: 10103 CPU
  [Created at cpu.304]
  Unique ID: rdCR.j8NaKXDZtZ6
  Hardware Class: cpu
  Arch: Intel
  Vendor: "GenuineIntel"
  Model: 6.15.13 "Intel(R) Pentium(R) Dual  CPU  T2390  @ 1.86GHz"
  Features:
fpu,vme,de,pse,tsc,msr,pae,mce,cx8,apic,sep,mtrr,pge,mca,cmov,pat,pse36,
clflush,dts,acpi,mmx,fxsr,sse,sse2,ss,ht,tm,pbe,nx,lm,constant_tsc,
arch_perfmon,pebs,bts,pni,dtes64,monitor,ds_cpl,est,tm2,ssse3,cx16,xtpr,
pdcm,lahf_lm
  Clock: 1866 MHz
  BogoMips: 3724.63
  Cache: 1024 kb
  Units/Processor: 2
  Config Status: cfg=new, avail=yes, need=no, active=unknown
```

7.4.3 Monitorizarea stării dispozitivelor hardware

Senzori

O altă sursă de informații despre hardware-ul prezent într-un sistem o reprezintă senzorii. Acești senzori pot oferi informații despre viteza ventilatoarelor, voltaje și temperatură, fiind folosite pentru a monitoriza starea sistemului.

Pachetul `lm-sensors` include programul **`sensors`** care afișează informații citite de la aceștia.

După instalarea pachetului trebuie rulată comanda **`sensors-detect`** care va detecta senzorii prezenți în sistem și va indica ce module trebuie încărcate. După ce configurarea se realizează cu succes, poate fi utilizată comanda **`sensors`**.

```
ubuntu@ubuntu:~$ sensors
smsc47m192-i2c-0-2d
Adapter: SMBus I801 adapter at 1400
+2.5V:      +2.60 V  (min =  +0.00 V, max =  +3.32 V)
VCore:      +1.42 V  (min =  +0.00 V, max =  +2.99 V)
+3.3V:      +3.13 V  (min =  +0.00 V, max =  +4.38 V)
+5V:        +5.16 V  (min =  +0.00 V, max =  +6.64 V)
[...]
smsc47m1-isa-0800
Adapter: ISA adapter
fan1:      2792 RPM  (min =  640 RPM, div = 8)
fan2:      2118 RPM  (min =  640 RPM, div = 8)
```

Pachetul de utilitare smartmontools

Pachetul `smartmontools` pune la dispoziție programe pentru monitorizarea hard-disk-urilor utilizând tehnologia S.M.A.R.T. (`Self-Monitoring, Analysis and Reporting Technology`). În unele sisteme, trebuie activat suportul S.M.A.R.T. în BIOS (unele BIOS-uri au această configurare în setul de opțiuni avansate). Tehnologia permite monitorizarea indicatorilor de stare a hard-disk-urilor în scopul anticipării posibilelor căderi.

În pachet se găsesc două programe:

- **`smartctl`** – pentru analiza stării S.M.A.R.T. la un moment dat;
- **`smartd`** – un daemon pentru monitorizarea permanentă.

Pentru a verifica starea la un moment dat pentru primul hard-disk, se poate utiliza comanda:

```
root@ubuntu:~# smartctl --all /dev/sda
smartctl version 5.37 [i686-pc-linux-gnu] Copyright (C) 2002-6 Bruce
Allen
Home page is http://smartmontools.sourceforge.net/

=== START OF INFORMATION SECTION ===
Model Family:     Seagate Barracuda 7200.7 and 7200.7 Plus family
Device Model:     ST3120827AS
Serial Number:    4MS03F43
Firmware Version: 3.42
User Capacity:    120,033,041,920 bytes
[...]

SMART Attributes Data Structure revision number: 10
Vendor Specific SMART Attributes with Thresholds:
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED
WHEN_FAILED RAW_VALUE
  1 Raw_Read_Error_Rate     0x000f   062   049   006    Pre-fail  Always
-       16
  3 Spin_Up_Time            0x0003   098   096   000    Pre-fail  Always
-       0
  4 Start_Stop_Count        0x0032   100   100   020    Old_age   Always
-       659
[...]
```

SMART are și suport pentru execuția de teste asupra discului. Există teste lungi și teste scurte. Execuția unui test poate fi urmărită în timp real dar poate fi și programată. Rezultatele testului se pot afla în ieșirea comenzii **`smartctl -all /dev/sda`**.

Un test scurt poate fi programat utilizând comanda:

```
root@ubuntu:~# smartctl --test=short /dev/sda
smartctl version 5.37 [i686-pc-linux-gnu] Copyright (C) 2002-6 Bruce
Allen
Home page is http://smartmontools.sourceforge.net/

=== START OF OFFLINE IMMEDIATE AND SELF-TEST SECTION ===
Sending command: "Execute SMART Short self-test routine immediately in
off-line mode".
Drive command "Execute SMART Short self-test routine immediately in
off-line mode" successful.
```

```
Testing has begun.
Please wait 1 minutes for test to complete.
Test will complete after Thu Sep 13 10:34:15 2007

Use smartctl -X to abort test.
```

Se poate observa la o nouă rulare a comenzii **`smartctl -all /dev/sda`** că apar la ieșire și rezultatele testului (dacă s-a terminat), precum în exemplul de mai jos.

```
root@ubuntu:~# smartctl --all /dev/sda
smartctl version 5.37 [i686-pc-linux-gnu] Copyright (C) 2002-6 Bruce
Allen
Home page is http://smartmontools.sourceforge.net/

=== START OF INFORMATION SECTION ===
[...]

=== START OF READ SMART DATA SECTION ===
SMART overall-health self-assessment test result: PASSED
[...]
```

Pe lângă testul scurt mai există și alte teste (spre exemplu un test lung și un test offline). Mai multe exemple de folosire a lui **`smartctl`** se pot afla din man smartctl și din **`smartctl -h`**.

Daemon-ul **`smartd`** monitorizează hard-disk-urile și permite înregistrarea stării acestora și a defectelor apărute prin intermediul `syslog`. În mod implicit verifică discurile la fiecare 30 minute. Daemon-ul se poate configura prin intermediul fișierului `/etc/smartd.conf`. Pentru lucrul cu daemoni se poate consulta capitolul 5.

7.5 Interfața cu dispozitivele din userspace – `udev` și `/dev`

În Linux există câte un fișier pentru fiecare dispozitiv în directorul `/dev`. Fișierele din `/dev` sunt create de managerul de dispozitive numit **`udev`**. Acesta este executat în întregime ca daemon în spațiul utilizator (user-space) și astfel poate executa programe arbitrare la apariția sau la dispariția dispozitivelor din sistem. **`udev`** oferă suport pentru nume persistente pentru dispozitive, ceea ce înseamnă că poate fi configurat să folosească întotdeauna același nume pentru fișierul asociat unui anumit dispozitiv (în implementările mai vechi pentru `/dev`, la reintroducerea în sistem a unor dispozitive, acestea puteau primi nume diferite comparativ cu introducerea anterioară).

`udev` primește notificări de la kernel atunci când apar sau dispar dispozitive. Astfel, spre exemplu, la conectarea unui USB stick în calculator, kernel-ul notifică **`udev`** că a apărut un nou dispozitiv de stocare. În funcție de schema de nume utilizată, **`udev`** poate, spre exemplu, să aleagă drept nume de dispozitiv următorul nume disponibil după numele hard-disk-ul prezent în sistem (presupunând că dispozitivul asociat hard-disk-ului este `/dev/sda`, udev alege ca nume `/dev/sdb`). După ce este ales un nume de dispozitiv, se poate executa un program care, în cazul prezentat aici, să monteze sistemul de fișiere de pe USB stick într-un subdirector din `/media`. Tot în funcție de configurări, anumite dispozitive pot să aibă fișierele asociate puse într-un subdirector separat din `/dev`.

Tabelul 7.2: Convenția de nume pentru dispozitive în Linux

Nume	Dispozitiv
`fd`	Floppy-disk
`hd`	Hard-disk IDE
`lp`	Imprimantă
`sd`	Hard-disk SATA (SCSI)
`tty`	Terminal fizic
`ram`	Dispozitiv care reprezintă memoria RAM
`loop`	Dispozitiv utilizat pentru montarea fișierelor precum dispozitivele reale

Pe lângă execuția de programe, **udev** mai permite modificarea permisiunilor pentru fișierele din `/dev` create. Astfel se poate configura, spre exemplu, ca pentru orice scanner introdus în sistem, fișierul creat să aparțină grupului cu numele scanner. În acest fel, numai utilizatorii care fac parte din acest grup vor putea lucra cu acest fișier și implicit cu scanner-ul. Trebuie amintit faptul că un utilizator poate să fie membru al mai multor grupuri.

Configurările pentru **udev** se găsesc în `/etc/udev/rules.d`. Acest director conține fișiere care descriu modul în care, în funcție de dispozitiv, se formează numele dispozitivelor, se dau permisiuni fișierelor sau se execută programe. De obicei acest director conține configurări aplicabile în orice situație și nu se recomandă editarea fișierelor din el.

7.6 Lucrul cu dispozitive (`/dev`)

S-a menționat anterior că **udev** se ocupă și de stabilirea permisiunilor pentru anumite dispozitive. Un utilizator trebuie să se găsească în grupul din care face fișierul dispozitivului pentru a putea utiliza dispozitivul (prin intermediul fișierului, bineînțeles).

Tabelul 7.2 prezintă câteva nume asociate în mod normal cu dispozitivele într-un mediu Linux. Trebuie menționat că de obicei numele sunt urmate de un număr sau o literă, care indică al câtelea dispozitiv de acel tip este în sistem. Spre exemplu, două unități de dischetă vor fi numite `/dev/fd0` și `/dev/fd1`, iar două hard-disk-uri vor fi numite `/dev/sda` și `/dev/sdb`.

Comanda următoare prezintă conținutul tipic pentru `/dev`:

```
1  ubuntu@ubuntu:~$ ls -l /dev/
2  total 0
3  lrwxrwxrwx 1 root root 10 2007-08-31 14:48 adsp -> sound/adsp
4  lrwxrwxrwx 1 root root 12 2007-08-31 14:48 agpgart -> misc/agpgart
5  lrwxrwxrwx 1 root root 11 2007-08-31 14:48 audio -> sound/audio
6  drwxr-xr-x 3 root root  0 2007-08-31 14:48 bus
7  drwxr-xr-x 2 root root  0 2007-08-31 14:48 cd
8  lrwxrwxrwx 1 root root 17 2007-08-31 14:48 cdrom -> /dev/cd/cdrom-sr0
9  lrwxrwxrwx 1 root root 17 2007-08-31 14:48 cdrom0-> /dev/cd/cdrom-sr0
10 [...]
11 drwxr-xr-x 2 root root  0 2007-08-31 14:48 loop
12 lrwxrwxrwx 1 root root  6 2007-08-31 14:48 loop0 -> loop/0
13 lrwxrwxrwx 1 root root  6 2007-08-31 14:48 loop1 -> loop/1
```

```
14 crw-r----- 1 root kmem  1,    1 2007-08-31 14:48 mem
15 lrwxrwxrwx 1 root root 11 2007-08-31 14:48 mixer -> sound/mixer
16 crw-rw-rw- 1 root root  1,    3 2007-08-31 14:48 null
17 [...]
18 crw-rw-rw- 1 root root  1,    8 2007-08-31 14:48 random
19 brw-rw---- 1 root disk  8,    0 2007-08-31 14:48 sda
20 brw-rw---- 1 root disk  8,    1 2007-08-31 14:48 sda1
21 brw-rw---- 1 root disk  8,    2 2007-08-31 14:48 sda2
22 brw-rw---- 1 root disk  8,   16 2007-08-31 14:48 sdb
23 [...]
```

7.6.1 Tipuri de dispozitive

În Linux, dispozitivele se împart în 2 categorii, fiecare categorie având particularitățile ei:

- dispozitive caracter
- dispozitive bloc

Dispozitivele pot fi diferențiate după litera care apare pe prima coloană în ieșirea comenzii **`ls -l`**, coloană care indică tipul fișierului. În lista de mai sus fișierele care au tipul `c` indică dispozitive de tip caracter iar fișierele cu tipul `b` indică dispozitive de tip bloc.

Câteva dispozitive sunt generate de kernel și au un comportament aparte. Aceste dispozitive sunt numite pseudo-dispozitive.

Dispozitivele caracter

> **Dispozitivele caracter** corespund dispozitivelor care transmit date câte un caracter odată.

Aceste dispozitive sunt de obicei folosite pentru a transmite fluxuri de date, precum:

- terminalul – `/dev/tty0`, `/dev/tty1`
- porturile seriale – `/dev/ttyS0`, `/dev/ttyS1`
- dispozitivele audio – `/dev/sequencer`, `/dev/mixer`, `/dev/midi`

Dispozitivele bloc

> Dispozitivele bloc corespund dispozitivelor care lucrează cu datele la nivel de bloc (scriu, citesc, șterg blocuri de date).

Dispozitivele care funcționează în acest fel sunt dispozitivele de stocare:

- fizice (hard disk-uri, unități CD-ROM/DVD-ROM etc.) – `/dev/sda`, `/dev/sda1`, `/dev/hda`
- virtuale (dispozitivele loop) – `/dev/loop0`

- asociate zonelor de memorie – `/dev/ram0`

Pseudo-dispozitive

Există o serie de dispozitive prezente în `/dev` care nu corespund unor componente hardware. Acestea sunt echivalente unor dispozitive virtuale care implementează comportamentul dispozitivelor normale de tip caracter.

Cele mai cunoscute sunt:

- `/dev/null` – fișier în care se poate scrie orice și totul este pierdut – de obicei ieșirile standard sau de eroare ale programelor sunt redirectate către `/dev/null` pentru a suprima afișarea oricărui mesaj;
- `/dev/zero` – fișier din care se poate citi la infinit octetul cu valoarea 0 – de obicei este folosit pentru a suprascrie zone de memorie/disc cu zerouri sau pentru a umple fișiere;
- `/dev/urandom` – fișier din care se pot citi la infinit octeți cu valori aleatoare – utilizat de obicei pentru a scrie fișiere sau zone de pe disc cu date aleatoare.

Câteva exemple de utilizare sunt prezentate mai jos.

Suprimarea ieșirii standard pentru un program:

```
1 ubuntu@ubuntu:~$ ./program > /dev/null
```

Suprimarea ieșirii de eroare pentru un program (mesajele de eroare se pierd):

```
1 ubuntu@ubuntu:~$ ./program 2>/dev/null
```

Suprimarea ambelor ieșiri (standard și eroare) pentru un program:

```
1 ubuntu@ubuntu:~$ ./program 2>&1 >/dev/null
```

Suprascrierea primei partiții de pe primul hard-disk cu zerouri:

```
1 root@ubuntu:~# dd if=/dev/zero of=/dev/sda1
```

Suprascrierea primului hard-disk cu date aleatoare:

```
1 root@ubuntu:~# dd if=/dev/urandom of=/dev/sda
```

7.6.2 Întreruperi hardware

> O întrerupere (**IRQ** – *interrupt request*) este un semnal trimis de un dispozitiv către procesor pentru a-l anunța că trebuie să-și oprească activitatea pentru a procesa un eveniment extern, cum ar fi apăsarea unei taste.

Pe platforma x86 sunt 16 IRQ-uri numerotate de la 0 la 15. Pe calculatoarele moderne pot exista mai mult de 16 întreruperi. Unele întreruperi sunt rezervate pentru dispozitive specifice, cum ar fi tastatura, iar altele pot fi reasignate. În tabelul 7.3 se află întreruperile și semnificația lor pentru platforma x86:

Tabelul 7.3: Semnificația întreruperilor

IRQ	Semnificație
0	System timer
1	Tastatura
2	Cascada pentru IRQ 8-15
3	Al doilea port serial (COM 2)
4	Primul port serial (COM 1)
5	Placa de sunet sau al doilea port paralel (LPT2)
6	Controler pentru Floppy
7	Primul port paralel (LPT1)
8	Real-time clock
9	Nerezervat
10	Nerezervat
11	Nerezervat
12	Mouse PS/2
13	Coprocesor aritmetic
14	Controler ATA primar
15	Controler ATA secundar

În cadrul sistemului de operare Linux, semnificația întreruperilor poate fi vizualizată folosind urmatoarea comandă:

```
ubuntu@ubuntu:~$  cat /proc/interrupts
           CPU0
  0:        130   IO-APIC-edge      timer
  1:        277   IO-APIC-edge      i8042
  6:          5   IO-APIC-edge      floppy
  7:          0   IO-APIC-edge      parport0
  8:          3   IO-APIC-edge      rtc
  9:          0   IO-APIC-fasteoi   acpi
 12:       1485   IO-APIC-edge      i8042
 14:          0   IO-APIC-edge      libata
 15:      13885   IO-APIC-edge      libata
 16:       4426   IO-APIC-fasteoi   eth1
 17:      43084   IO-APIC-fasteoi   ioc0
NMI:          0
LOC:     254957
ERR:          0
MIS:          0
```

În comanda anterioară, pentru fiecare întrerupere este afișat driver-ul care o folosește.

7.6.3 Adrese I/O

Adresele I/O, numite și porturi I/O, sunt locații unice în memorie rezervate pentru comunicarea între procesor și diversele dispozitive fizice. În tabelul 7.4 se poate observa corespondența între denumirea dispozitivului pe Linux și pe Windows și adresa I/O asociată:

În Linux se pot verifica adresele I/O folosind următoarea comandă:

Tabelul 7.4: Adresele I/O ale dispozitivelor

Denumire Linux	Denumire Windows	IRQ asociat	Adresa I/O
/dev/ttyS0	COM1	4	0x03f8
/dev/ttyS1	COM2	3	0x02f8
/dev/ttyS2	COM3	4	0x03e8
/dev/ttyS3	COM4	3	0x02e8
/dev/lp0	LPT1	7	0x0378-0x037f
/dev/lp1	LPT2	5	0x0278-0x027f

```
ubuntu@ubuntu:~$ cat /proc/ioports
0000-001f : dma1
0020-0021 : pic1
0040-0043 : timer0
0050-0053 : timer1
0060-006f : keyboard
0070-0077 : rtc
0080-008f : dma page reg
00a0-00a1 : pic2
00c0-00df : dma2
00f0-00ff : fpu
```

7.6.4 Adrese DMA

DMA (*Direct Memory Access*) se referă la adresarea directă a memoriei și este o alternativă la adresele I/O. DMA face posibil transferul datelor între dispozitive și memorie fără ajutorul procesorului. Acest lucru duce la îmbunătățirea performanțelor sistemului deoarece procesorul nu mai este ocupat cu operații I/O.

Arhitectura x86 implementează acest concept folosind canale DMA, câte unul pentru fiecare dispozitiv. Următoarea comandă va afișa ce canale DMA sunt folosite de sistem:

```
ubuntu@ubuntu:~$ cat /proc/dma
 2: floppy
 4: cascade
```

7.6.5 Manipularea datelor la nivel scăzut (comanda dd)

Pentru a lucra cu datele la nivel de octet sistemele de operare de tip Unix pun la dispoziție comanda **dd**. Această comandă poate fi utilizată pentru a manevra date de orice dimensiune. Se bazează pe faptul că datele sunt transferate de la o sursă la o destinație, eventual cu simple transformări.

Comanda primește sursa și destinația sub forma a doi parametri:

- **sursa**: `if=<fișier_intrare>` (dacă lipsește acest parametru se presupune că sursa este intrarea standard)
- **destinația**: `of=<fișier_ieșire>` (dacă lipsește acest parametru se presupune că destinația este ieșirea standard)

Fișierele pot fi atât fișiere normale cât și fișiere care reprezintă dispozitive bloc sau dispozitive caracter. Astfel, o utilizare foarte simplă a comenzii **dd** este următoarea:

```
ubuntu@ubuntu:~$ dd if=/dev/zero of="myfile"
```

Comanda de mai sus copiază de la intrare la ieșire octeți până când apare o eroare, din moment ce nu a fost specificată nicio limită. În aceste condiții, comanda se termină doar dacă fișierul sursă e limitat sau fișierul destinație e limitat. În cazul de mai sus, sursa are dimensiune infinită iar destinația e limitată doar de dimensiunea spațiului liber disponibil pe partiția unde se scrie fișierul.

Pentru a specifica dimensiunea care se dorește a fi copiată se utilizează o combinație a următorilor parametri:

- **numărul de blocuri**: `count=<număr_blocuri>` – dacă acest parametru lipsește copierea se oprește doar în condițiile prezentate mai sus
- **dimensiunea unui bloc**: `bs=<dimensiune_în_octeți>` (*block size*) – pot fi folosiți și multiplii, spre exemplu: `bs=1048576`, `bs=1024K`, `bs=1M` sunt toate echivalente; dacă lipsește acest parametru se utilizează dimensiunea implicită a unui bloc de 512 octeți (dimensiunile blocurilor citite și dimensiunile blocurilor scrise pot fi ajustate individual prin utilizarea parametrilor `ibs` și `obs`; ele sunt modificate simultan prin utilizarea parametrului `bs`)

Exemplele următoare ilustrează utilizarea comenzii dd împreună cu parametrii precizați anterior:

- copierea de 10 MB de date din 10 blocuri de 1 MB (1024K); copierea se face din `/dev/zero` către `/dev/null` deci nu are niciun efect:

```
ubuntu@ubuntu:~$ dd if=/dev/zero of=/dev/null bs=1024K count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 0.00490124 seconds, 2.1 GB/s
```

- același efect ca mai sus doar că se precizează M în loc de K:

```
ubuntu@ubuntu:~$ dd if=/dev/zero of=/dev/null bs=1M count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 0.00509437 seconds, 2.1 GB/s
```

- copierea de 10MB de date în blocuri de 1M la intrare și 128K la ieșire:

```
ubuntu@ubuntu:~$ dd if=/dev/zero of=/dev/null count=10 ibs=1M obs
=128K
10+0 records in
80+0 records out
10485760 bytes (10 MB) copied, 0.00622419 seconds, 1.7 GB/s
```

Comanda **dd** permite și poziționarea în fișierele utilizate pentru intrare și ieșire:

- intrare: `skip=<număr_blocuri>` – sare `<număr_blocuri>` din fișierul de intrare, echivalentul a `<număr_blocuri> * ibs` octeți;
- ieșire: `seek=<număr_blocuri>` – sare `<număr_blocuri>` din fișierul de ieșire, echivalentul a `<număr_blocuri> * obs` octeți.

Tabelul 7.5: Informații expuse prin intermediul lui /proc

Fișier	Conținut
`/proc/cmdline`	Linia care este folosită la inițializarea kernel-ului, inclusiv parametrii trimiși acestuia
`/proc/cpuinfo`	Informații despre procesorul/procesoarele din sistem
`/proc/devices`	Lista dispozitivelor caracter și a celor bloc
`/proc/interrupts`	Statistici despre numărul de întreruperi și dispozitivele asociate întreruperilor
`/proc/meminfo`	Informații despre utilizarea memoriei
`/proc/modules`	Lista modulelor încărcate la un moment dat în sistem – comanda **`lsmod`** prezintă conținutul fișierului într-un mod user-friendly
`/proc/partitions`	Lista partițiilor
`/proc/swaps`	Lista dispozitivelor pe care se face swap
`/proc/uptime`	Timpul trecut de la ultima pornire a sistemului, în secunde – comanda **`uptime`** prezintă conținutul fișierului într-un mod user-friendly
`/proc/version`	Versiunea kernel-ului care rulează la un moment dat

Poziționarea este utilă în cazul în care se dorește ca de pe dispozitivul de intrare să se citească o anumită zonă, precum un anumit sector de pe hard disk. Exemplul următor ilustrează acest lucru – se sare peste primul sector și se citesc două sectoare de pe hard disk (trebuie menționat că dimensiunea unui sector pe HDD este 512 octeți, valoarea implicită pentru parametrul `bs`):

```
1 root@ubuntu:~# dd if=/dev/sda of=/dev/null skip=1 count=2
2 2+0 records in
3 2+0 records out
4 1024 bytes (1.0 kB) copied, 0.043731 seconds, 23.4 kB/s
```

7.7 Analiza sistemului

7.7.1 Informații despre sistem

Prin intermediul directorului `/proc`, kernel-ul expune și o serie de informații legate de sistem. Tabelul 7.5 următor prezintă fișierele mai importante din `/proc` și conținutul acestora.

Conținutul pentru oricare din fișierele prezentate mai sus se poate afla folosind comanda **`cat`**. De exemplu, folosind comanda **`cat`** cu argumentul `/proc/cpuinfo` se vor afișa următoarele informații:

```
1 ubuntu@ubuntu:~$ cat /proc/cpuinfo
2 processor       : 0
3 vendor_id       : GenuineIntel
4 cpu family      : 6
5 model           : 15
6 model name      : Intel(R) Core(TM)2 Duo CPU     T7500  @ 2.20GHz
7 stepping        : 8
```

```
cpu MHz         : 2194.893
cache size      : 4096 KB
fdiv_bug        : no
[...]
```

Folosind comanda anterioară s-au obținut următoarele informații: `processor` reprezintă identificatorul procesorului prezentat (pe sistemele cu un singur procesor, va apărea doar procesorul cu identificatorul 0), `cpu family` reprezintă un număr care se va apenda la numărul 86 pentru a obține numele familiei de procesoare (în exemplul de față 686), `model name` reprezintă denumirea procesorului, `cpu MHz` reprezintă frecvența procesorului iar `cache size` este dimensiunea memoriei cache L2.

Pentru a obține informații despre memoria utilizată de sistem, se va afișa conținutul fișierului `/proc/meminfo`:

```
ubuntu@ubuntu:~$ cat /proc/meminfo
MemTotal:       255676 kB
MemFree:          9684 kB
Buffers:         10668 kB
Cached:          84448 kB
SwapCached:       3004 kB
Active:         179812 kB
Inactive:        43268 kB
SwapTotal:      401400 kB
SwapFree:       360748 kB
[...]
```

Folosind comanda anterioară s-au afișat următoarele informații: `MemTotal` este cantitatea totală de RAM, `MemFree` este cantitatea de RAM care nu este utilizată de sistem, `Buffers` este cantitatea de RAM folosită de buffere, `Cached` este cantitatea de RAM folosită ca memorie cache, `SwapCached` este cantitatea de memorie swap folosită ca memorie cache, `Active` este cantitatea de buffere sau memorie cache care este în curs de utilizare, `Inactive` este cantitatea de buffere sau memorie cache care este liberă, `SwapTotal` este cantitatea totală de swap disponibilă și `SwapFree` este cantitatea de swap liberă.

Folosind următoarea comandă se vor afișa: versiunea kernelului Linux, a compilatorului **gcc** și a distribuției de Linux folosite:

```
ubuntu@ubuntu:~$ cat /proc/version
Linux version 2.6.24-24-generic (buildd@rothera) (gcc version 4.2.4 (
Ubuntu
4.2.4-1ubuntu3)) #1 SMP Tue Aug 18 17:04:53 UTC 2009
```

Oricare dintre aceste fișiere poate fi folosit pentru a găsi, la un moment dat, o informație despre sistem. Spre exemplu, dacă se dorește aflarea tipului procesorului/procesoarelor din sistem se poate utiliza comanda următoare:

```
ubuntu@ubuntu:~$ cat /proc/cpuinfo | grep "model name"
model name      : Intel(R) Core(TM)2 Duo CPU     T7500  @ 2.20GHz
```

Este important de reținut că `/proc` expune informații care sunt accesibile și prin intermediul unor comenzi. Aceste informații, fie că sunt citite direct din `/proc`, fie că sunt citite prin intermediul comenzilor, pot fi filtrate pentru a se extrage date utile pentru utilizator.

7.7.2 Kernel "tunables"

`sysctl` este o interfață care permite configurarea unor parametri din kernel în timpul execuției acestuia. Parametrii pot fi modificați în două feluri:

- prin modificarea directă a fișierelor din `/proc/sys`;
- prin utilizarea comenzii **`sysctl`**.

Primul caz presupune scrierea și citirea parametrilor din anumite căi, precum în exemplul următor:

```
root@ubuntu:~# cat /proc/sys/net/ipv4/ip_forward
0

root@ubuntu:~# echo "1" > /proc/sys/net/ipv4/ip_forward

root@ubuntu:~# cat /proc/sys/net/ipv4/ip_forward
1
```

Comenzile de mai sus au activat rutarea pachetelor de rețea la nivelul kernel-ului, scriind "1" în `/proc/sys/net/ipv4/ip_forward`. Această facilitate este utilă dacă se dorește ca un sistem care are acces la Internet să permită accesul unei întregi subrețele.

Folosind comanda **`sysctl`**, același lucru se obține în felul următor:

```
root@ubuntu:~# sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 0

root@ubuntu:~# sysctl -w net.ipv4.ip_forward="1"
net.ipv4.ip_forward = 1
```

Se observă că pentru a afișa starea unui parametru se folosește o adresare asemănătoare cu cea a structurilor din C/C++. Comanda a doua modifică un parametru prin utilizarea opțiunii `--w` (write).

Modificările prezentate anterior sunt modificări temporare, ele fiind pierdute în cazul în care se repornește sistemul. Pentru a efectua configurări permanente, acestea se scriu în fișierul `/etc/sysctl.conf`. Presupunând că se dorește dezactivarea forwarding-ului, sintaxa în fișier este destul de simplă (liniile care încep cu `#` din fișier sunt comentarii):

```
ubuntu@ubuntu:~$ cat /etc/sysctl.conf
[...]
# Controls IP packet forwarding
net.ipv4.ip_forward = 1
[...]
```

Fișierul de configurare este citit la fiecare pornire a sistemului. Pentru a forța citirea fișierului de configurare, se poate utiliza comanda **`sysctl -p`**:

```
root@ubuntu:~# sysctl -p
net.ipv4.ip_forward = 1
```

7.8 Studii de caz

7.8.1 Salvarea și restaurarea MBR și a tabelei extinse de partiții

MBR-ul (*Master Boot Record*) reține informații despre partițiile primare aflate pe un hard-disk precum atribute ale acestora, poziția lor pe disc și dimensiunea lor. MBR-ul mai conține și bootloader-ul – un mic program care pornește un sistem de operare prezent pe computer. Într-un mediu mixt Windows/Linux, la o instalare a Windows-ului se va observa că Linux-ul devine inaccesibil (bootloader-ul Windows-ului nu cunoaște nimic despre Linux). În aceste condiții, pentru a preîntâmpina acest lucru, se recomandă realizarea unui back-up pentru MBR. Mai multe detalii despre MBR găsiți în secțiunea 6.2.2.

Backup-ul MBR-ului de pe primul HDD se realizează cu următoarea comandă:

```
root@ubuntu:~# dd if=/dev/sda of=mbr.img bs=512 count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 4.8004e-05 seconds, 10.7 MB/s
```

În urma execuției comenzii se realizează fișierul `mbr.img` de 512 octeți cu MBR-ul. Acest fișier poate fi plasat într-un loc sigur și accesibil în cazul în care discul devine inaccesibil (pe un USB stick sau CD-ROM spre exemplu).

Pentru a restaura MBR-ul, se efectuează operația inversă:

```
root@ubuntu:~# dd if=mbr.img of=/dev/null bs=512 count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 4.622e-05 seconds, 11.1 MB/s
```

În MBR se poate stabili ca una dintre partiții să fie partiție extinsă. În partiția extinsă pot fi create mai multe partiții logice, pentru o mai bună organizare a discului. Informațiile despre partițiile logice nu sunt ținute în MBR, deoarece nu încap în cei 512 octeți. Partițiile logice sunt reținute sub formă de listă înlănțuită în afara MBR-ului.

În momentul în care se lucrează cu schema de partiții de pe un hard disk, se recomandă realizarea unui backup înainte. Modificarea schemei de partiții nu șterge datele de pe hard disk, doar schimbă descrierea partițiilor (locul unde se găsesc acestea). În situația în care se șterge o partiție importantă (fie ea logică sau primară), este util să existe un backup al acestor informații pentru recuperare. Modul de backup pentru MBR a fost descris anterior.

Pentru a realiza backup-ul configurației tuturor partițiilor se utilizează programul **`sfdisk`**. Acesta analizează atât partițiile primare cât și pe cele logice, dar nu se uită la informațiile din MBR care nu sunt legate de partiționare (precum bootloader). El este un bun complement pe lângă metoda amintită mai sus de a face backup la MBR.

Comanda următoare face backup pentru întreaga schemă de partiționare folosind **`sfdisk`** – fișierul de backup este `/usbstick/partitions.backup`:

```
root@ubuntu:~# sfdisk -d /dev/sda > /usbstick/partitions.backup
```

Pentru a restaura schema de partiționare salvată anterior, **`sfdisk`** se utilizează astfel:

```
root@ubuntu:~# sfdisk /dev/sda < /usbstick/partitions.backup
```

7.8.2 Salvarea conținutului unui disc cu sectoare inaccesibile

Există situații când, de obicei după o perioada mai lungă de utilizare, un hard disk începe să nu mai funcționeze corect. Se observă acest lucru din erorile pe care sistemul de operare le afișează când vrea să citească date de pe disc. În asemenea situații, cea mai rapidă metodă de backup este copierea fișierelor pe un alt disc.

Acesta este cazul favorabil, când datele mai pot fi identificate și sistemul de operare mai funcționează. În cazul în care defectul hard disk-ului apare în zona în care este descris sistemul de fișiere, există o probabilitate destul de mare ca datele să nu mai poată fi salvate prin simpla copiere a fișierelor. În această situație se poate opta pentru repararea sistemului de fișiere utilizând aplicația **`fsck`**, dar nu se dorește ca această reparare să aibă loc pe hard disk-ul defect. Întâi se realizează o copie a tuturor datelor de pe primul hard disk pe un alt hard disk, copie numită imagine.

Comanda utilizată pentru a realiza această imagine este dd, împreună cu parametrul `conv=noerror`, parametru care îi spune dd-ului să nu se oprească atunci când detectează un bloc inaccesibil.

```
root@ubuntu:~# dd if=/dev/sda of=/dev/sdb conv=noerror
```

Execuția acestei comenzi poate dura foarte mult timp, datorită numărului mare de blocuri care trebuie citite și scrise. Pentru a accelera execuția comenzii, se poate mări dimensiunea blocului citit/scris prin intermediul parametrului `bs` (block size), prezentat anterior. Utilizarea unui bloc mai mare decât dimensiunea unui sector de hard disc (512 octeți) poate duce la pierderea ultimelor sectoare de pe discul sursă, căci nu se poate realiza un bloc cu ele.

7.8.3 Crearea unei imagini de CD; montarea unei imagini

O imagine de CD se creează în mod asemănător creării unei imagini de hard disk.

```
root@ubuntu:~# dd if=/dev/hda of=cd.iso
```

Această comandă va citi CD-ul introdus în unitatea `/dev/hda` sector cu sector și va scrie imaginea lui în fișierul `cd.iso`.

Pentru a putea utiliza imaginea, aceasta trebuie montată într-un director precum orice alt sistem de fișiere, dar cu următoarele observații:

- tipul sistemului de fișiere de pe un CD este `iso9660`;
- deoarece fișierul nu este un dispozitiv real, trebuie utilizată și opțiunea `loop`.

```
root@ubuntu:~# mount -t iso9660 -o loop cd.iso /mnt/cd
```

Pentru a demonta imaginea, se utilizează comanda:

```
root@ubuntu:~# umount /mnt/cd
```

În acest fel pot fi păstrate pe disc imagini ale CD-urilor utilizate des și/sau pot fi testate imagini de CD create de diferite utilitare.

Trebuie menționat că se poate aplica aceeași procedură și pentru DVD-uri, atâta timp cât sistemul de fișiere este corect precizat în comanda **mount** (DVD-urile pot fi scrise în formatul UDF).

7.8.4 Utilizarea unui fișier de pe o partiție FAT32 ca fișier de swap pentru un LiveCD Linux

Se observă de multe ori că LiveCD-urile funcționează greu pe sisteme care nu au multă memorie RAM disponibilă. Într-un sistem instalat, memoria RAM poate fi ajutată și de prezența spațiului de swap de pe hard disk. Într-un sistem în care nu există nicio partiție de swap, un LiveCD nu activează niciun astfel de spațiu. O posibilitate o reprezintă utilizarea unui fișier de swap, pe o partiție deja existentă în sistem. S-a utilizat pentru exemplificare o partiție FAT32, dar se poate utiliza orice tip de partiție pe care Linux-ul poate scrie.

Pentru început, se montează sistemul de fișiere (partiția a doua, primul hard disk) unde se dorește crearea fișierului de swap:

```
1 root@ubuntu:~# mount -t vfat /dev/sda2 /mnt/temp
```

În acest moment se poate crea un fișier de swap de dimensiunea dorită utilizând programul **dd**:

```
1 root@ubuntu:~# dd if=/dev/zero of=/mnt/temp/fisier.swap bs=1M count=256
2 256+0 records in
3 256+0 records out
4 268435456 bytes (268 MB) copied, 6.41171 seconds, 41.9 MB/s
```

S-a creat un fișier de 256 MB (plin cu zerouri) pentru a fi folosit ca fișier de swap. În acest moment trebuie transformat fișierul în zonă de swap, printr-un proces asemănător formatării partițiilor:

```
1 root@ubuntu:~# mkswap /mnt/temp/fisier.swap
```

În acest moment, fișierul este pregătit pentru a fi folosit ca zonă de swap. Sistemul de operare trebuie să afle de prezența fișierului de swap printr-o operație asemănătoare montării unui sistem de fișiere:

```
1 root@ubuntu:~# swapon /mnt/temp/fisier.swap
```

După execuția acestei comenzi sistemul folosește fișierul creat mai sus ca zonă de swap.

Cuvinte cheie

- kernel
- kernel-space
- user-space
- mașină virtuală
- von Neumann
- CPU, procesor

- RAM, DRAM
- placă de bază
- chipset
- northbridge
- southbridge
- hard-disk, HDD
- magistrală, PCI, AGP
- dispozitiv
- modul
- driver
- lsmod, insmod, modprobe, rmmod
- /sys
- lsusb, lspci
- sensors
- smartmontools
- /dev
- udev
- dispozitive de tip bloc
- dispozitive de tip caracter
- pseudodispozitive
- dd
- /proc
- sysctl
- /etc/sysctl.conf
- MBR

Întrebări

1. Care dintre următoarele NU este o componentă a arhitecturii von Neumann?
   - ❑ CPU
   - ❑ unitatea de comandă
   - ❑ memoria
   - ❑ dispozitivele de intrare ieșire
2. Care utilitar poate fi folosit pentru efectuarea unui backup la MBR?
   - ❑ **`lspci`**
   - ❑ **`lsusb`**
   - ❑ **`sysctl`**
   - ❑ **`dd`**
3. Care din următoarele NU oferă direct informații despre sistem?
   - ❑ `/sys`
   - ❑ `/proc`
   - ❑ **`dd`**
   - ❑ **`lsusb`**
4. Care comandă NU este folosită pentru interacțiunea cu modulele din kernel?
   - ❑ **`lsmod`**
   - ❑ **`modprobe`**

- ❑ **modinstall**
- ❑ **insmod**

5. Care din următoarele NU este un exemplu de pseudodispozitiv?
   - ❑ `/dev/null`
   - ❑ `/dev/zero`
   - ❑ `/dev/hda3`
   - ❑ `/dev/urandom`
6. Care din următoarele dispozitive NU este asociat unui hard-disk?
   - ❑ `/dev/sda1`
   - ❑ `/dev/hda5`
   - ❑ `/dev/sda`
   - ❑ `/dev/ttyS0`
7. Care din următoarele se referă în mod direct la placa de bază?
   - ❑ southbridge
   - ❑ megabyte
   - ❑ GPU
   - ❑ cache
8. Care din următoarele acronime NU este corelat direct cu dispozitive fizice?
   - ❑ SCSI
   - ❑ USB
   - ❑ ACPI
   - ❑ TGZ
9. La care dispozitiv se referă acronimul AGP?
   - ❑ procesor
   - ❑ placa de rețea
   - ❑ placa grafică
   - ❑ hard-disk
10. La ce se referă în general noțiunea de "arhitectură pe N biți"?
    - ❑ dimensiunea registrelor
    - ❑ dimensiunea magistralei de date
    - ❑ dimensiunea memoriei cache L1
    - ❑ viteza maximă a southbridge-ului

Capitolul 8

Configurări de rețea

You know it's love when you memorize her IP address to skip DNS overhead.

Ce se învață din acest capitol?

- Noțiunea de rețea de calculatoare
- Topologia unei rețele de calculatoare
- Adresarea IP
- Parametrii unei rețele de calculatoare: adresă IP, mască de rețea, gateway
- Configurarea unei interfețe de rețea în Linux; adăugarea de rute în Linux
- Configurare temporară; configurare permanentă; configurare statică; configurare dinamică
- Configurare DNS
- Verificarea conectivității într-o rețea locală
- Configurarea rețelei în Windows
- Configurarea PPPoE în Linux

8.1 Concepte de rețea

Apariția și evoluția calculatoarelor au dus la dorința de a facilita comunicația între ele pentru a partaja date mai ușor sau pentru a putea îndeplini o sarcină comună. Deși dezvoltarea rețelelor de calculatoare a fost inițial lentă, implementarea lor a cunoscut o creștere mare după 1990, odată cu explozia numărului de calculatoare legate la Internet.

Dacă inițial rețelele de calculatoare au fost create pentru a facilita schimbul de informație în mediul academic, în momentul de față ele se adresează în mare măsură utilizatorilor privați și mediului comercial.

Pentru ca un calculator să fie conectat la o rețea trebuie realizate două conexiuni: o conexiune fizică și o conexiune logică.

Conexiunea fizică este utilizată pentru a codifica informația sub formă de semnale electrice (în cazul cel mai uzual întâlnit conexiunea se realizează prin cablu UTP), semnale optice (pentru legăturile prin fibra optică) sau semnale electromagnetice (pentru legăturile fără fir). În primele două cazuri ea se realizează printr-un cablu care leagă calculatorul de un echipament de rețea.

Conexiunea logică cuprinde un set de mecanisme de adresare complexe (ce asigură comunicația între calculatoarele conectate la rețea), cuprinde controlul fluxului de date și oferă garanția integrității datelor. Cel mai cunoscut mecanism de adresare prezent în cadrul conexiunii logice este IP (Internet Protocol). Acesta asigură identificarea unică și comunicarea calculatoarelor atât în rețelele locale, cât și în Internet.

8.1.1 Noțiuni de bază

Rețelele de calculatoare, din punctul de vedere al standardelor folosite, sunt împărțite în trei mari categorii: rețele locale de calculatoare (*Local Area Network* – LAN), rețele metropolitane de calculatoare (*Metropolitan Area Network* – MAN) și rețele de calculatoare pe arii extinse (*Wide Area Network* – WAN). În momentul de față această clasificare nu se mai păstrează 100%, rețelele MAN dispărând aproape complet și fiind înlocuite cu rețele LAN.

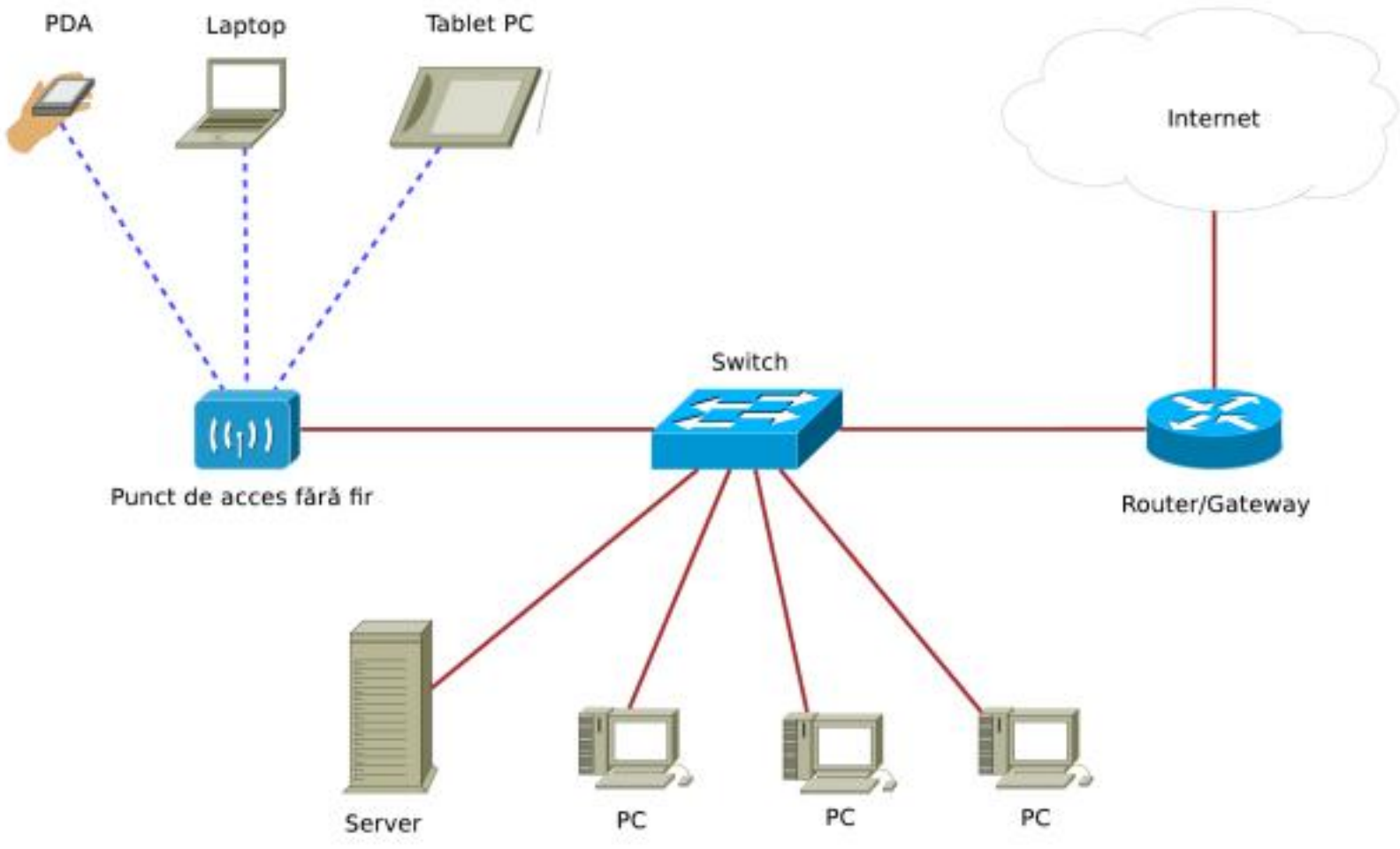


Figura 8.1: Schema unei rețele locale de calculatoare

O rețea locală de calculatoare este formată cel mai adesea din mai multe stații legate între ele. O stație poate fi un calculator, un laptop, un PDA, un *Tablet PC*, un server, un router etc.

Conectarea staților se realizează printr-un echipament de rețea numit switch. Acesta transferă informația între calculatoare bazându-se pe adresa MAC.

> **Adresa MAC** este un număr cu o lungime de 48 de biți care identifică în mod unic fiecare interfață de rețea Ethernet.

Prin urmare nu pot exista două interfețe de rețea Ethernet cu aceeași adresă MAC. Adresa MAC este înscrisă din fabrică într-o memorie ROM de pe placă și ea nu poate fi schimbată.

Datorită faptului că protocolul DHCP de configurare automată a adreselor IP utilizează frecvent adresa MAC pentru a aloca o adresă IP unui calculator, este uzuală practica de a modifica adresa MAC a unei plăci de rețea. Acest lucru nu înseamnă rescrierea memoriei ROM, ci specificarea unei alte adrese pe care sistemul de operare să o folosească în locul celei inițiale.

Switch-ul permite comunicația între calculatoare aflate în aceeași rețea locală. Pentru a putea accesa și alte calculatoare (cel mai adesea pentru a putea accesa Internetul) trebuie ca unul din echipamentele din rețea să aibă o a doua legătură la o altă rețea. Acest echipament poate să fie o stație sau un server cu mai multe interfețe de rețea sau un echipament de rețea dedicat numit **ruter**. Transferul informației din rețeaua locală mai departe înspre Internet se bazează pe adresa IP.

> **Adresa IP** este un număr cu o lungime de 32 de biți folosit pentru identificarea fiecărei interfațe de rețea în Internet.

O stație poate avea mai multe adrese IP, în mod uzual câte o adresă pentru fiecare interfață. Este posibil însă ca pe o interfață să fie configurate mai multe adrese IP.

8.1.2 IPv6

Până acum am folosit noțiunea IP pentru a ne referi la Internet Protocol version 4 (IPv4). Internet Protocol version 6 (IPv6) este următoarea generație de protocol de rețea, dezvoltat pentru a înlocui treptat IPv4.

> **Adresa IPv6** este un număr cu o lungime de 128 de biți folosit pentru identificarea fiecărei interfațe de rețea în Internet.

Principalul motiv pentru care a fost implementat un nou protocol de rețea pentru Internet este epuizarea adreselor IPv4. IPv6 are un spațiu de adrese mult mai mare decât IPv4 din cauza lungimii adreselor. Spațiul de adrese IPv6 oferă 2^128 adrese, față de IPv4 care are doar 2^32 adrese.

8.1.3 Moduri de adresare: adresare unicast, multicast, broadcast

Comunicațiile între echipamente pot fi împărțite în trei categorii, în funcție de numărul de destinatari:

- comunicații **unicast**: au o sursă și un destinatar;
- comunicații **multicast**: au o sursă și mai mulți destinatari ce fac parte dintr-un grup specific;
- comunicații **broadcast**: au o sursă și ca destinatar toate stațiile dintr-o rețea.

Atât mesajele de tip multicast cât și cele de tip broadcast sunt adresate mai multor destinatari simultan. Diferența dintre ele constă în faptul că mesajele de tip broadcast se adresează tuturor stațiilor dintr-o rețea, pe când cele multicast se adresează doar unora din aceste stații.

8.2 Parametri de rețea

8.2.1 Adresă IP și mască de rețea

Adresa IP pe 32 de biți a fost introdusă în perioada '80, când nu se preconiza creșterea spectaculoasă a Internetului. După 1990, datorită dezvoltării rețelelor, cerințele pentru adrese IP au crescut, astfel încât nu peste mult timp adresele au început să se epuizeze. Din acest motiv au apărut mai multe mecanisme menite să reducă numărul de adrese IP utilizate, printre care adresele IP private și translatarea de adrese IP (Network Address Translation – NAT). În paralel a fost dezvoltată și o soluție pe termen lung: protocolul IPv6, a cărui utilizare va crește în anii următori. În cadrul acestui protocol adresa IP are 128 de biți.

Pentru a ușura utilizarea adreselor IP, cei 32 de biți sunt separați în patru grupuri a câte 8 biți, fiecare grup fiind scris în formă zecimală. Astfel adresa IP utilizată în mod uzual este compusă din patru numere zecimale între 0 și 255 despărțite prin puncte:

```
01011001110101110001111010000111 - cei 32 de biti ai unei adrese IP
01011001.11010111.00011110.10000111 - patru grupuri a cate 8 biti
89.215.30.135 - fiecare grup este transformat in format zecimal
```

Atunci când o interfață de rețea are configurată o adresă IP, adresa are două componente: o parte din cei 32 de biți reprezintă adresa rețelei din care face parte stația și cealaltă parte a biților reprezintă adresa stației în cadrul rețelei. Sistemul este similar cu cel utilizat pentru codul poștal: în cadrul codului 014288, primele două cifre pot reprezenta județul, următoarele două pot reprezenta orașul, iar ultimele două pot reprezenta strada. Astfel destinația este localizată în arii din ce în ce mai restrânse.

Pentru a putea spune care dintre cei 32 de biți reprezintă adresa rețelei și care reprezintă adresa calculatorului în cadrul rețelei, este utilizată o mască de rețea. Aceasta este tot un număr de 32 de biți cu o caracteristică specială: este compusă, pornind de la stânga la dreapta, dintr-un bloc compact de 1 urmat de un bloc compact de 0. Pentru utilizarea mai ușoară a măștii de rețea este folosită aceeași notație ca și în cadrul adresei IP:

```
11111111000000000000000000000000 - cei 32 de biti ai unei masti
11111111.00000000.00000000.00000000 - patru grupuri a cate 8 biti
255.0.0.0 - fiecare grup este tranformat in format zecimal
```

Dacă în cadrul măștii de rețea un bit este egal cu 1, atunci bitul corespunzător din adresa IP face parte din *adresa rețelei*, iar dacă este egal cu 0, bitul corespunzător din adresa IP face parte din adresa calculatorului (stației) *în cadrul rețelei*:

```
01011001.11010111.00011110.10000111 - adresa IP
11111111.00000000.00000000.00000000 - masca de retea
<-A.R.-> <---------A.S.------------>
A.R.- Adresa retelei
A.S.- Adresa statiei
```

Pentru a putea afla din ce rețea face parte o adresă IP, se face ŞI logic pe biți între adresa IP și masca de rețea. După realizarea operației de ŞI logic se obține o adresa IP care are toți biții din partea de adresă a rețelei egali cu adresa rețelei din care face parte stația, și toți biții din partea de adresa a stației în cadrul rețelei egali cu 0:

```
01011001.11010111.00011110.10000111 = 89.215.30.135 - adresa IP
11111111.00000000.00000000.00000000 = 255.0.0.0 - masca de retea
----------------------------------- - SI logic pe biti
01011001.00000000.00000000.00000000 = 89.0.0.0 - adresa retelei
<-A.R.-> <-------A. C.------------>
01011001.11111111.11111111.11111111 = 89.255.255.255 - adr. broadcast
```

Adresa IP în care toți biții din partea de adresă a stației sunt egali cu 0 se numește **adresa rețelei** din care face parte stația. Toate stațiile dintr-o rețea locală au adrese IP care au aceeași adresă a rețelei. În felul acesta se realizează corespondența între conexiunea fizică la rețea și cea logică.

Adresa IP în care toți biții din partea de adresă a stației în cadrul rețelei sunt egali cu 1 se numește **adresa de broadcast** a rețelei. Dacă un pachet este trimis către această adresă, atunci el va fi procesat de toate calculatoarele din rețea.

Clase de adrese IP

Conceptul de clase de adrese IP este important pentru înțelegerea modului în care funcționează o rețea de calculatoare. Spațiul de adrese IP este împărțit în cinci categorii, denumite *clase de adrese*. În funcție de intervalul din care face parte primul octet al adresei IP (sau, echivalent, primul număr zecimal din notația cu punct), se stabilește clasa respectivei adrese IP:

- între 0 și 127, adresa IP face parte din clasa A,
- între 128 și 191, adresa IP face parte din clasa B,
- între 192 și 223, adresa IP face parte din clasa C,
- între 224 și 239, adresa IP face parte din clasa D,
- între 240 și 255, adresa IP face parte din clasa E.

Această împărțire a claselor de adrese a fost aleasă din considerente de performanță: pentru a determina clasa unei adrese IP oarecare, un dispozitiv hardware de rețea trebuie să inspecteze maximum primii patru biți ai adresei IP (valoarea biților notați cu - nu este importantă):

- 0 - - -. - - - -, adresa IP face parte din clasa A,
- 10 - -. - - - -, adresa IP face parte din clasa B,
- 110 -. - - - -, adresa IP face parte din clasa C,

- 1110. - - - -, adresa IP face parte din clasa D,
- 1111. - - - -, adresa IP face parte din clasa E.

Împărțirea pe clase determină două caracteristici importante ale unei adrese: tipul rutării (unicast/multicast/rezervat) și masca de rețea.

- Clasa A: 0.0.0.0 – 127.255.255.255 – masca 255.0.0.0 – unicast
- Clasa B: 128.0.0.0 – 191.255.255.255 – masca 255.255.0.0 – unicast
- Clasa C: 192.0.0.0 – 223.255.255.255 – masca 255.255.255.0 – unicast
- Clasa D: 224.0.0.0 – 239.255.255.255 – multicast
- Clasa E: 240.0.0.0 – 255.255.255.255 – rezervat

Modul acesta de clasificare a spațiului de adresă a fost utilizat într-o fază incipientă a Internetului. Din motive care țin de eficiența modului de utilizare a adreselor IP, în prezent, măștile de rețea nu au doar aceste trei lungimi fixe, ci pot avea lungime variabilă, iar împărțirea în cele cinci clase nu mai este întânită decât în cazuri rare.

Adrese IP publice și adrese IP private

În momentul în care numărul de adrese IP rămase libere a început să scadă simțitor, la începutul anilor '90, au apărut mecanisme menite să rezolve parțial această problema. Printre mecanismele apărute se numără și spațiul de adrese private.

Din fiecare clasă de adrese a fost rezervat un spațiu de adrese care nu pot fi accesate direct din afara rețelei locale. Practic stațiile care au configurate adrese IP private sunt invizibile din afara rețelei lor, și implicit din Internet. De aceea pot exista mai multe stații, făcând parte din rețele diferite, care au aceeași adresa IP privată. În felul acesta se realizează o economie mare de adrese IP.

Spațiile de adrese private sunt:

```
10.0.0.0 -- 10.255.255.255 din clasa A
170.16.0.0 -- 172.31.255.255 din clasa B
192.168.0.0 -- 192.168.255.255 din clasa C
```

Pentru ca o stație ce are configurată o adresă IP privată să poată accesa stații care nu se află în aceeași rețea cu ea este necesară utilizarea mecanismelor de NAT (*Network Address Translation*). De cele mai multe ori, prin NAT, toate stațiile dintr-o rețea cu adrese IP private vor fi recunoscute în afara rețelei ca având aceeași adresă IP publică.

8.2.2 Ruter implicit (default gateway)

Pentru a putea accesa calculatoare aflate în alte rețele, este necesar ca unul dintre echipamentele conectate în rețea să aibă o a doua interfață conectată la o altă rețea. Din cea de-a doua rețea se poate realiza mai departe accesul către Internet. Acest echipament se numește *gateway*.

Atunci când o stație A dorește să comunice cu o stație B, primul pas pe care îl face este să verifice dacă adresa IP a lui B este în aceeași rețea cu adresa lui A. În cazul în care

cele două stații sunt în aceeași rețea, informațiile sunt trimise direct către B. Dacă B nu se află în aceeași rețea cu A, atunci informațiile sunt trimise către gateway, urmând ca acesta să găsească o rută către B.

De exemplu când A1 vrea să comunice cu A3, fiind amandouă în aceeași rețea locală, îi va trimite informațiile direct. Calea pachetelor este reprezentată în figura 8.2.

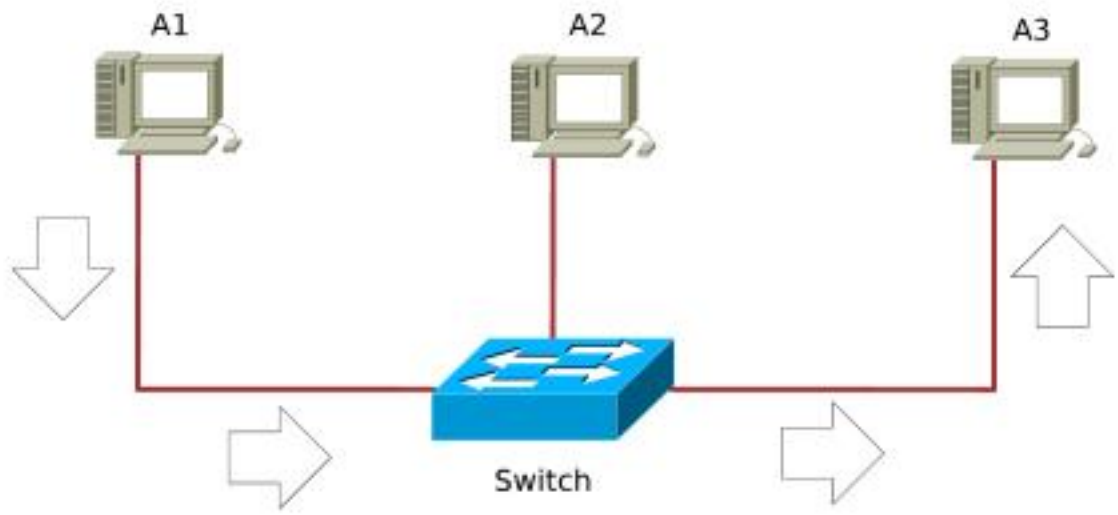


Figura 8.2: Comunicarea în rețeaua locală

În cazul în care A1 vrea să comunice cu B2, va trimite informațiile către R1 care este gateway-ul. R1 va verifica dacă rețeaua destinație este direct conectată sau dacă cunoaște calea către ea. Schema din figura 8.3 reprezintă calea pachetelor de la A1 la B2:

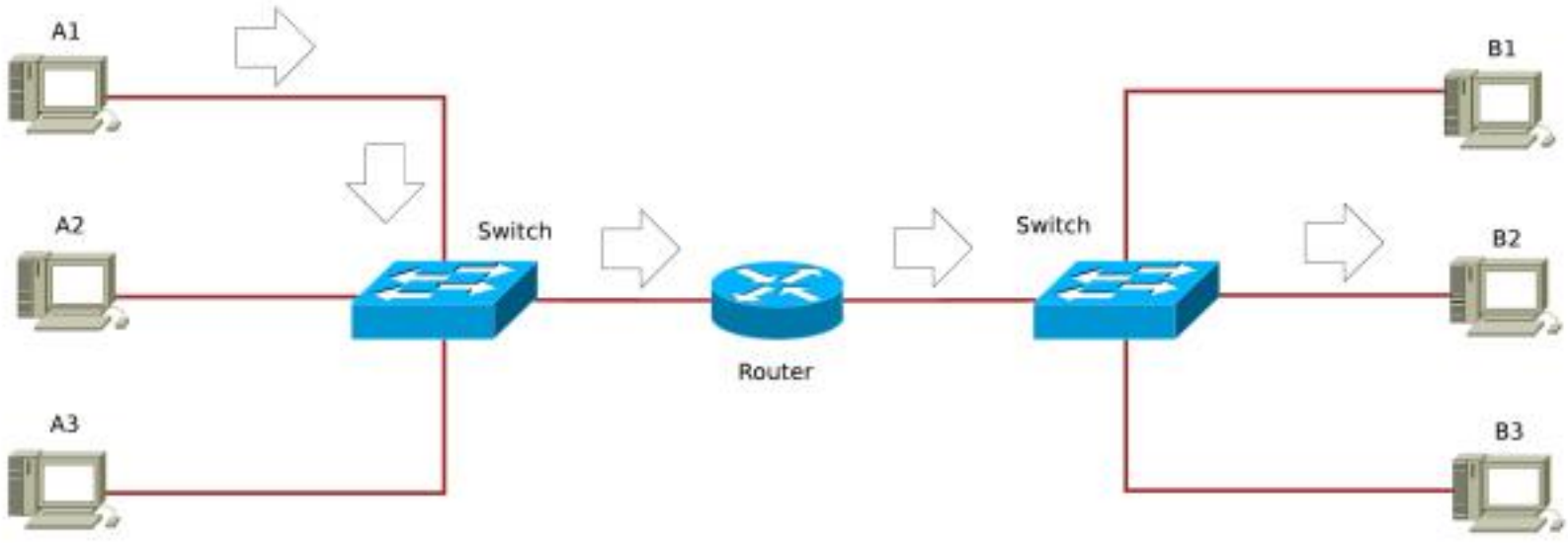


Figura 8.3: Comunicarea în afara rețelei locale

Prin urmare este necesar ca fiecare stație din rețea să cunoască adresa gateway-ului pentru a putea avea acces în afara rețelei și masca de rețea pentru a vedea dacă este cazul să folosească gateway-ul sau nu.

8.2.3 DNS

Un utilizator accesează în mod normal un număr mare de servicii de rețea, cele mai multe fiind oferite de diverse servere. Accesarea fiecărui serviciu este condiționată de cunoașterea adresei IP a serverului care oferă serviciul respectiv. Spre exemplu, pentru a accesa o pagină web este necesară cunoașterea adresei IP a serverului pe care este găzduită acea pagină.

Pentru că memorarea unui număr mare de adrese IP nu este un lucru comod, a apărut serviciul numit Sistemul numelor de domenii (Domain Name System – DNS). În cadrul acestui serviciu se realizează corespondența între un șir de caractere și o adresă IP. De aceea DNS-ul poate fi privit ca o carte de telefon de unde, dacă se cunoaște numele unui server, i se poate afla adresa IP.

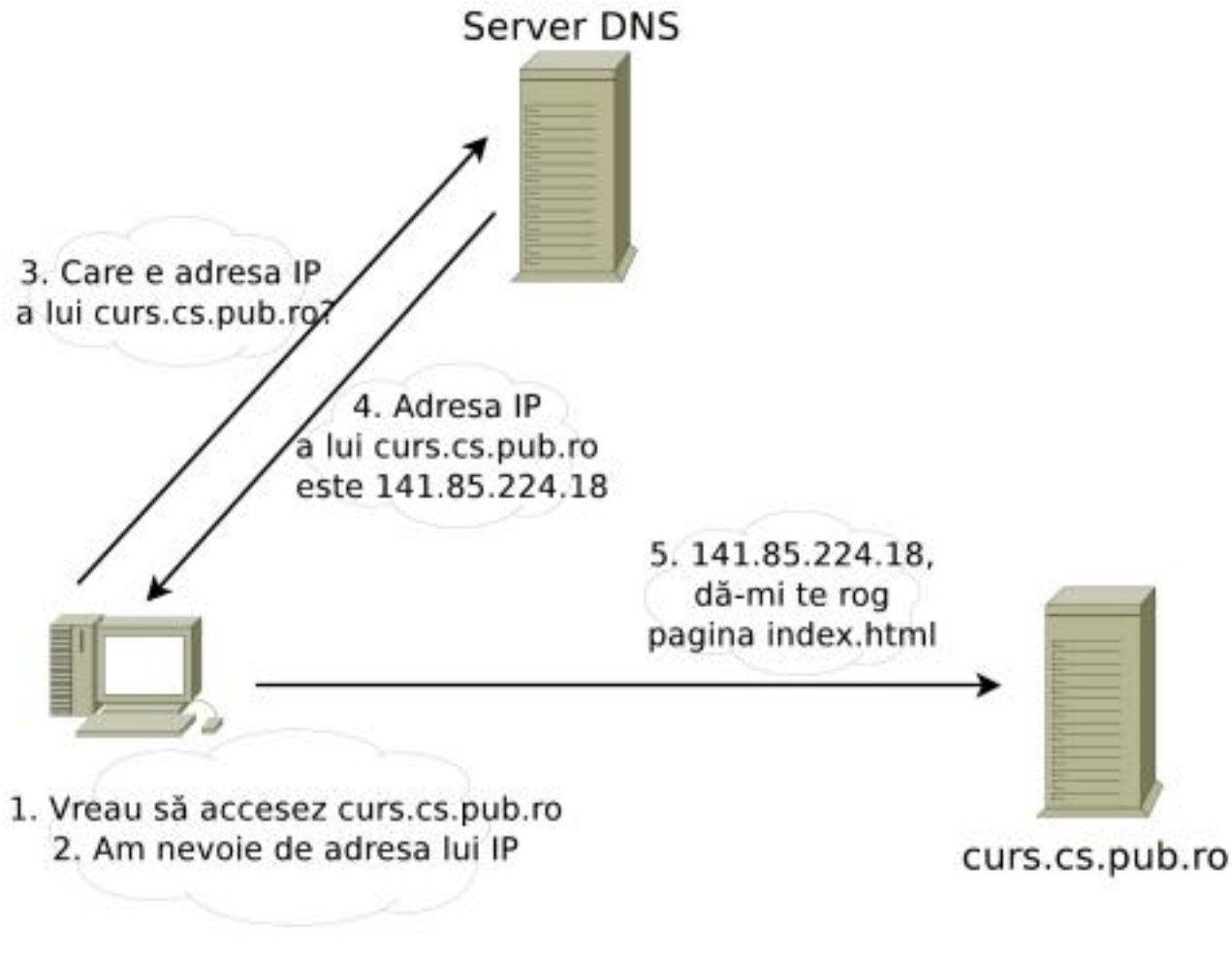


Figura 8.4: Rolul unui server DNS

Un nume de domeniu (asociat unui calculator) este compus din mai multe părți despărțite prin puncte: de exemplu curs.cs.pub.ro. Cea din extremitatea dreaptă se numește *Top Level Domain – TLD* și poate conține indicativul unei țări (.ro pentru România, .us pentru SUA, .jp pentru Japonia etc.) sau poate reprezenta un anumit tip de organizație (de exemplu .com pentru organizațiile comerciale, .org pentru organizațiile non-profit, .mil pentru armata SUA etc).

În stânga Top Level Domain se găsesc subdomenii. În exemplul de mai sus subdomeniul pub este asociat Universității Politehnica București, cs este asociat Facultății de Automatică și Calculatoare, Catedra de Calculatoare.

Serviciile de DNS sunt oferite de servere specializate. Pentru a putea accesa un astfel de server, este necesară cunoașterea adresei lui IP. Exista servere publice care oferă servicii de DNS, însă de cele mai multe ori este de preferat ca în cadrul rețelei locale să existe un server de DNS din motive ce țin de timpul de răspuns.

Chiar dacă serverul de DNS este plasat în rețeaua locală, DNS-ul rămâne cel mai lent serviciu de rețea și componenta cu cel mai mare timp de răspuns în mecanismul de comunicație în rețea. De aceea se spune că "*You know it's love when you memorize her IP address to skip DNS overhead*".

Este posibil uneori ca adresa IP asociată unei stații să fie dinamică. Aceasta înseamnă că din timp în timp stația respectivă va primi o altă adresă IP. Pentru a putea accesa stația fără să i se cunoască adresa IP existentă la un moment dat se folosește sistemul

Tabelul 8.1: Configurarea rețelei

	Configurare statică	**Configurare dinamică**
Configurare temporară	**`ifconfig`**	**`dhclient`, `dhcpcd`**
Configurare permanentă	`/etc/network/interfaces`	`/etc/network/interfaces`

numit DynDNS (*Dynamic DNS*). În cadrul acestui sistem stația va trimite către un server DNS specializat informații actualizate legate de adresa ei IP.

8.3 Configurări temporare

8.3.1 Interfețe de rețea. Configurări permanente și configurări temporare

Din punct de vedere logic, fiecărei plăci de rețea îi corespunde în cadrul sistemului de operare o interfață. Pe această interfață se configurează o adresă IP și o mască de rețea.

În Linux interfețele de rețea conțin în denumirea lor două componente. Prima parte reprezintă tipul interfeței (de exemplu `eth` pentru Ethernet, `wlan` pentru placile de rețea fără fir etc) și numărul ei (`eth0` reprezintă prima interfață de Ethernet, `wlan1` reprezintă a două interfață fără fir etc).

Configurarea unei adrese IP se poate realiza în două moduri: temporară sau permanentă. Fiecare din cele două se poate realiza la rândul ei static (parametrii sunt introduși manual de administrator) sau dinamic (parametrii se configurează în mod automat).

Configurarea temporară statică se realizează cu ajutorul comenzii **`ifconfig`** (a cărei denumire vine de la *interface configuration*). Efectul acestei configurări este imediat (parametrii se aplica imediat după ce comanda este dată) însă odată cu repornirea sistemului de operare, configurările se pierd.

Configurarea temporară dinamică se realizează cu ajutorul unui client DHCP, de exemplu **`dhclient`** sau **`dhcpcd`**.

Configurarea permanentă statică sau dinamică se realizează cu ajutorul fișierului `/etc/network/interfaces`. Efectul acestei comenzi nu este imediat (configurările nu se aplică odată cu salvarea fișierului), însă configurările se păstrează și după repornirea sistemului de operare. Acest lucru se petrece deoarece fișierul în care sunt trecute configurările este citit și interpretat de sistemul de operare la inițializare, configurările din fișier fiind din nou aplicate.

Şi configurarea unui gateway se poate realiza în mod temporar cu ajutorul comenzii route sau în mod permanent prin fișierul de configurare `/etc/network/interfaces`.

Fiecare din cele două metode (permanent vs. temporar) are avantajele sale. Configurările temporare au efect imediat și se aplică ușor. Cele permanente se pot

aplica automat la fiecare inițializare a sistemului de operare.

8.3.2 Configurarea temporară statică a unei adrese IP pe o interfață

Pentru a configura o interfață de rețea în mod temporar se folosește comanda **`ifconfig`**. Modul de utilizare al comenzii este următorul:

```
root@ubuntu:~# ifconfig [-v] [-a] [-s] [interface]
root@ubuntu:~# ifconfig [-v] interface [aftype] options | address ...
```

Prin utilizarea comenzii fără niciun parametru se pot afla informații despre interfețele de rețea active:

```
root@asgard:/home/george# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          inet6 addr: fe80::214:d1ff:fe38:73a6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500 Metric:1
          RX packets:18982017 errors:0 dropped:0 overruns:0 frame:0
          TX packets:27864548 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:144959484 (130 MiB) TX bytes:21665362 (20.6 MiB)
          Interrupt:18 Base address:0x4400

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:16436 Metric:1
          RX packets:5777 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5777 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:360919 (352.4 KiB) TX bytes:360919 (352.4 KiB)
```

Pe lângă interfețele corespunzătoare plăcilor de rețea există și interfața `lo`. Numele acestei interfețe vine de la *loopback*. Orice informație transmisă pe loopback se va întoarce înapoi tot pe loopback.

Interfața de loopback este o interfață virtuală (fără un corespondent fizic) care este folosită în scopuri de testare. Ea are, deobicei, asociată o adresă IP standard: `127.0.0.1`. Dacă un calculator nu are nicio placă de rețea, interfața de loopback va fi singura existentă.

Parametrii uzuali ai comenzii **`ifconfig`** sunt următorii:

- `-v` afișează informații detaliate în cazul erorilor;
- `-a` afișează informații despre toate interfețele existente, fie că sunt active sau inactive.

```
root@asgard:/home/george# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]

```

```
root@asgard:/home/george# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
eth1      Link encap:Ethernet  HWaddr 00:18:F3:AB:45:62
          BROADCAST MULTICAST  MTU:1500  Metric:1
          [...]
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]
```

- `-s` afișează statistici legate de interfețele de rețea în format compact, sub forma unui tabel:

```
root@asgard:/home/george# ifconfig -s
iface   MTU Met  RX-OK RX-ERR  [...]  TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0   1500 0   193488      0  [...] 282347      0      0      0 BMRU
lo    16436 0     5849      0  [...]    584      0      0      0 LRU
```

- `interface` reprezintă numele interfeței despre care se doresc informații sau numele interfeței care va fi configurată. De exemplu, pentru a afișa informații despre interfața de rețea eth0 se folosește comanda

```
root@asgard:/home/george# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
```

- `address` reprezintă adresa IP care va fi configurată pe interfață. De exemplu, pentru a configura adresa `IP 10.1.1.3` pe interfața `eth0`:

```
root@asgard:/home/george# ifconfig eth0 10.1.1.3

root@asgard:/home/george# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.3  Bcast:10.255.255.255  Mask:255.0.0.0
          [...]
```

- `options` reprezintă opțiuni avansate ce pot fi configurate

Opțiunile cele mai des folosite sunt următoarele:

- `down` dezactivează o interfață. Odată dezactivată, interfața nu va mai trimite și nu va mai primi niciun pachet

```
root@asgard:/home/george# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]

root@asgard:/home/george# ifconfig eth0 down

root@asgard:/home/george# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]
```

Se poate observa faptul că, după ce interfața a fost dezactivată, ea nu mai apare în lista de interfețe active. Pentru a afișa toate interfețele existente se folosește parametrul `-a`.

- `up` este utilizată pentru a activa o interfață. În cazul în care este specificată o adresă IP pentru a fi configurată pe interfață, opțiunea `up` este considerată implicit.

```
root@asgard:/home/george# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]

root@asgard:/home/george# ifconfig eth0 up

root@asgard:/home/george# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]
```

- `netmask addr` specifică masca de rețea asociata interfeței. În cazul în care această opțiune nu este prezentă se va considera masca implicită clasei IP-uri din care face parte adresa IP a interfeței Se poate observa din exemplele anterioare faptul că, nespecificând masca de rețea, aceasta a fost considerată implicit `255.0.0.0`, care este valoarea pentru clasa A de adrese IP din care face parte `10.1.1.3`. În cazul în care se specifică și valoarea măștii ca opțiune, rezultatul va fi:

```
root@asgard:/home/george# ifconfig eth0 10.1.1.3 netmask
255.255.255.0

root@asgard:/home/george# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.3  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
```

- `hw class addr` specifică utilizarea unei alte adrese MAC și nu cea din memoria ROM a plăcii de rețea, în cazul în care driverul plăcii de rețea oferă suport pentru acest lucru. Este foarte important ca interfața de rețea să fie dezactivată înaintea configurării unei noi adrese MAC:

```
root@asgard:/home/george# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.3  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]

root@asgard:/home/george# ifconfig eth0 down

root@asgard:/home/george# ifconfig eth0 hw ether 01:02:03:04:05:06
up

root@asgard:/home/george# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 01:02:03:04:05:06
          inet addr:10.1.1.3  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
```

Utilitatea folosirii unei adrese MAC diferită de cea a plăcii de rețea apare datorită faptului că protocolul DHCP de configurare automată a adreselor IP utilizează frecvent adresa MAC pentru a aloca o adresă IP unui calculator. Pentru a obține o anumită adresă IP, trebuie folosită adresa MAC asociată în cadrul serverului DHCP cu adresa IP dorită.

8.3.3 Asigurarea conectivității la Internet. Configurarea temporară statică a unei rute implicite

Comanda **`ifconfig`** permite doar configurarea unei adrese IP pe o interfață. Pentru a putea accesa Internetul este nevoie să fie specificată adresa unui gateway.

Specificarea temporară a unui gateway se realizează cu ajutorul unei rute. O **rută** reprezintă calea pe care o urmează pachetele în drumul lor către destinație. Rutele sunt reținute în cadrul unui structuri numite **tabelă de rutare**. Aceasta conține practic informații despre toate destinațiile accesibile la un moment dat.

Printre informațiile reținute pentru fiecare rută se numără: adresa destinației (poate fi adresa unei stații sau adresa unei întregi rețele), masca de rețea asociată și adresa echipamentului de rețea către care se trimit informațiile pentru a ajunge la destinație.

Configurarea tabelei de rutare se realizează cu ajutorul comenzii route. Aceasta permite adăugarea, modificarea și ștergerea de rute. Pentru a vedea care sunt rutele existente la un moment dat (ca atare pentru a vedea conținutul tabelei de rutare) se utilizează comanda **`route`** fără parametri:

```
root@asgard:/home/george# route
Kernel IP routing table
Destination   Gateway      Genmask         Flags Metric Ref  Use Iface
localnet      *            255.255.255.0   U     0      0      0 eth0
```

În rezultatul de mai sus se vede faptul că singura rută existentă este cea asociată rețelei locale (`localnet`) .Această rută (care este practic asociată cu rețeaua locală direct conectată) este introdusă în tabela de rutare după ce interfața de rețea asociată rețelei este configurată.

Formatul comenzii utilizat pentru adăugarea unei rute implicite este următorul:

```
root@@ubuntu:~# route add default gateway addr
```

Configurarea unei rute implicite se realizează astfel:

```
root@asgard:/home/george# route add default gateway 10.1.1.1

root@asgard:/home/george# route
Kernel IP routing table
Destination   Gateway      Genmask         Flags Metric Ref  Use Iface
localnet      *            255.255.255.0   U     0      0      0 eth0
default       10.1.1.1     0.0.0.0         UG    0      0      0 eth0
```

Unul dintre parametrii utili ai comenzii route este `-n` (*numeric*) Folosind acest parametru adresele din tabela de rutare vor fi afișat în format numeric în loc să se afișeze numele serverelor sau stațiilor. Este folosit atunci când se dorește interogarea unui server DNS (de exemplu atunci când nu este configurat un astfel de server).

```
root@asgard:/home/george# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.1.0        *               255.255.255.0   U     0      0        0 eth0
0.0.0.0         10.1.1.1        0.0.0.0         UG    0      0        0 eth0
```

8.4 Configurări permanente

8.4.1 Configurarea permanentă a unei interfețe de rețea

Configurările de rețea permanente se realizează în cadrul fișierului `/etc/network/interfaces`. Manualul de utilizare al acestui fișier se poate accesa cu comanda:

```
man interfaces
```

Un exemplu de fișier de configurare este următorul:

```
root@asgard:/home/george# cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo eth0
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
iface eth0 inet static
        address 10.1.1.2
        netmask 255.255.255.0
        network 10.1.1.0
        broadcast 10.1.1.255
        gateway 10.1.1.1
        dns-nameservers 10.1.1.1
```

Fișierul `/etc/network/interfaces` conține diverse configurări ale interfețelor de rețea, configurări care sunt interpretate de utilitarele **`ifup`** și **`ifdown`**. **`ifup`** este un utilitar ce permite activarea unei interfețe de rețea. El este similar opțiunii up a comenzii **`ifconfig`**. **`ifdown`** dezactivează o interfață de rețea, fiind similar cu opțiunea `down` a comenzii **`ifconfig`**.

Pentru a activa toate interfețele definite cu `auto` în `/etc/network/interfaces` se folosește următoarea comandă:

```
root@ubuntu:~# ifup -a
```

Pentru a activa o anumită interfață se precizează numele acesteia, de exemplu:

```
root@ubuntu:~# ifup eth0
```

Pentru a dezactiva toate interfețele active se folosește următoarea comandă:

```
root@ubuntu:~# ifdown -a
```

În cadrul fișierului `interfaces` se găsesc mai multe declarații de tipul `iface`, `auto` sau `allow`.

Declarațiile `auto` specifică ce interfețe sunt activate atunci când **`ifup`** este folosit împreună cu parametrul `-a`. Un astfel de apel (**`ifup -a`**) este folosit la pornirea sistemului de operare. Pornirea unei interfețe marcate ca `auto` este realizată în momentul în care `udev` (vezi secțiunea 7.5) a creat mecanismele necesare funcționării interfeței.

Declarațiile `allow` specifică interfețele care sunt activate automat de către diverse subsisteme. Astfel activarea nu mai este realizată după ce `udev` a detectat placa de rețea, ci în momentul în care aceasta devine disponibilă.

Pentru specificarea unei interfețe de loopback se folosește următoarea linie:

```
1 iface lo inet loopback
```

După declarația `iface` urmează numele interfeței, apoi tipul de adrese pe care interfața îl folosește (`inet` reprezintă adresare IP, `inet6` reprezintă adresare IPv6). Pentru a marca o interfață ca loopback, se folosește parametrul `loopback`.

O linie asemănătoare este folosită și pentru a specifică orice alta interfață existentă. Spre exemplu, pentru configurarea interfeței `eth0`:

```
1 iface eth0 inet static
```

După declarația `iface` urmează numele interfeței apoi tipul de adrese pe care îl folosește interfața, iar la final, prin includerea parametrului `static` se specifică faptul că interfața este configurată manual de administrator, rămânând fixă până la următoarea schimbare manuală. O altă metodă de configurare este cea în care se folosește DHCP, prin care stația cere configurările interfeței de la un server DHCP.

Dacă parametrul `static` a fost precizat în declarația `iface`, atunci după această declarație trebuie să urmeze parametrii cu care va fi configurată interfața de rețea:

```
1 iface eth0 inet static
2         address 10.1.1.2
3         netmask 255.255.255.0
4         network 10.1.1.0
5         broadcast 10.1.1.255
6         gateway 10.1.1.1
7         dns-nameservers 10.1.1.1
```

`address` specifică adresa IP a interfeței (parametru obligatoriu), `netmask` specifică masca de rețea care va fi folosită (parametru obligatoriu), `network` specifica adresa rețelei din care face parte interfața (parametru obligatoriu doar pentru kernelul 2.0.x, adresă ce se poate obține făcând ŞI logic pe biți între adresa IP și masca de rețea, `broadcast` specifică adresa de broadcast a rețelei din care face parte interfața (parametru opțional).

După modificarea fișierului `/etc/network/interfaces` este necesară restartarea serviciului de rețea pentru a citi noua configurație:

```
1 root@ubuntu:~# /etc/init.d/networking restart
```

Serviciul de rețea poate fi oprit folosind argumentul `stop` și pornit folosind argumentul `start`:

```
root@ubuntu:~# /etc/init.d/networking stop

root@ubuntu:~# /etc/init.d/networking start
```

8.4.2 Configurarea permanentă statică a unei rute implicite

O a doua modalitate de specificare a rutei implicite este folosirea fișierului `/etc/network/ interfaces`. Structura acestuia a fost descrisă în paragraful anterior.

Printre parametrii care se pot configura după declarația `iface` se află și gateway-ul implicit. Specificarea acestuia se realizează astfel:

```
iface eth0 inet static
        [...]
        gateway 10.1.1.1
```

unde `10.1.1.1` este adresa IP a gateway-ului.

8.4.3 DHCP. Configurarea unei interfețe în mod automat

Configurarea parametrilor unei interfețe de rețea se poate realiza static sau dinamic.

În cazul configurărilor statice toți parametrii trebuie introduși manual pe fiecare stație din rețea. În cazul în care se schimbă gateway-ul, de exemplu, pe fiecare stație din rețea trebuie realizată această schimbare.

Configurările dinamice permit atribuirea automată a parametrilor de rețea, printre care adresa IP, masca de rețea, gateway-ul, serverul de DNS.

Cel mai utilizat protocol de configurare dinamică (automată) este **DHCP** (*Dynamic Host Configuration Protocol*). Pentru ca acest protocol să funcționeze este necesar ca în rețeaua locală să se găsească un server de DHCP. Serverului îi sunt specificate plaja de adrese IP pe care le poate atribui calculatoarelor și restul configurărilor de rețea.

Atunci când o stație dorește să obțină o adresă IP, va trimite un mesaj către toată rețeaua (broadcast) prin care încearcă să determine dacă există un server de DHCP. Dacă acest server există, atunci el va trimite configurațiile către stație.

Configurarea parametrilor interfeței de rețea folosind DHCP se poate realiza în mod temporar sau permanent.

Pentru a realiza o configurare temporară se pornește clientul de DHCP, de exemplu cu ajutorul comenzii **`dhclient`**:

```
root@asgard:/home/george# dhclient
Internet Systems Consortium DHCP Client V3.0.6
Copyright 2004-2007 Internet Systems Consortium.
All rights reserved.
For info, please visit http://www.isc.org/sw/dhcp/

Listening on LPF/eth0/00:40:f4:cc:c5:41
Sending on   LPF/eth0/00:40:f4:cc:c5:41
```

```
Sending on   Socket/fallback
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 6
DHCPOFFER from 10.1.1.1
DHCPREQUEST on eth0 to 255.255.255.255 port 67
DHCPACK from 10.1.1.1
bound to 10.1.1.2 -- renewal in 815 seconds.
```

Interfața va rămâne astfel configurată până la următoarea repornire a stației sau până când configurările vor fi schimbate.

Pentru a realiza o configurare permanentă prin DHCP se folosește de asemenea fișierul `/etc/network/interfaces`. Linia care configurează interfața eth0 arată astfel:

```
iface eth0 inet dhcp
```

8.4.4 DNS. Configurarea serverelor de DNS

Fișiere de configurare

Pentru a configura adresele serverelor de DNS se folosește fișierul `/etc/resolv.conf`. În cadrul acestui fișier se pot specifica adresele mai multor servere de DNS. În cazul în care unul din ele nu conține informații despre domeniul căutat, se va încerca interogarea celorlalte.

Manualul de utilizare al fișierului `/etc/resolv.conf` se poate accesa folosind comanda:

```
man resolv.conf
```

Un exemplu de fișier `/etc/resolv.conf` este următorul:

```
search localdomain
nameserver 10.1.1.1
nameserver 217.115.138.24
nameserver 128.107.241.185
```

În cadrul fișierului cea mai importantă declarație este declarația nameserver. Ea definește adresa unui server de DNS. Fișierul poate conține mai multe declarații de acest tip.

Aplicarea configurărilor se realizează imediat deoarece fișierul `resolv.conf` este interogat de fiecare proces atunci când dorește rezolvarea unui nume de domeniu.

Deși nu este uzual, serverele de DNS se pot configura și din cadrul fișierului `/etc/network/interfaces`.

De cele mai multe ori, atunci când o interfață este configurată automat prin DHCP, serverul de DHCP oferă pe lângă adresa IP și adresa unui server DNS, care va fi salvată automat în `/etc/resolv.conf`.

Utilitarul `resolvconf`

Folosind utilitarul **`resolvconf`** se pot afișa informații despre serverele de nume configurate în sistem și se poate modifica configurația din fișierul

/etc/resolv.conf. Folosind opțiunea -a se poate modifica fișierul /etc/resolv.conf:

```
ubuntu@ubuntu:~$ cat /etc/resolv.conf
nameserver 192.168.2.1

ubuntu@ubuntu:~$ echo "nameserver 192.168.2.100" | resolvconf -a eth0

ubuntu@ubuntu:~$ echo "nameserver 192.168.2.101" | resolvconf -a eth1

ubuntu@ubuntu:~$ cat /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by
resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 192.168.2.100
nameserver 192.168.2.101
```

O diferență între configurarea cu **resolvconf** și modificarea directă a fișierului /etc/ resolv.conf este specificarea interfeței pentru care se face această configurație. Pentru a șterge configurația asociată cu o anumită interfață se folosește următoarea comandă:

```
ubuntu@ubuntu:~$ resolvconf -d eth0

ubuntu@ubuntu:~$ cat /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by
resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 192.168.2.101
```

Interogarea serverelor de nume

Folosind comanda **host** se pot obține informații despre un anumit domeniu, cum ar fi adresa IP asociată, serverul de nume al domeniului, serverul de mail, descrierea, alias-ul etc.

De exemplu, adresa IP asociată se poate afla folosind următoarea comandă:

```
ubuntu@ubuntu:~$ host -t A cs.pub.ro
cs.pub.ro has address 141.85.37.5
```

Pentru a obține serverul de nume asociat cu acest domeniu se va folosi comanda:

```
ubuntu@ubuntu:~$ host -t NS cs.pub.ro
cs.pub.ro name server ns.cs.pub.ro.
cs.pub.ro name server pub.pub.ro.
```

Serverul de mail se poate obține folosind comanda următoare:

```
ubuntu@ubuntu:~$ host -t MX cs.pub.ro
cs.pub.ro mail is handled by 5 mail.cs.pub.ro.
```

Comanda următoare va afișa descrierea domeniului:

```
ubuntu@ubuntu:~$ host -t TXT cs.pub.ro
cs.pub.ro descriptive text "Politehnica University of Bucharest,
Computer Science Departament"
```

8.4.5 Aliasuri. /etc/hosts

Aliasurile reprezintă etichete care sunt asociate unor adrese de IP. Sistemul de aliasuri funcționează asemănător cu cel de DNS: de exemplu, atunci când o stație dorește să acceseze serverul numit `my-print-server`, ea va încerca să găsească adresa IP a serverului respectiv.

Există însă două deosebiri majore față de DNS. Prima constă în faptul ca etichetele asociate unor adrese IP nu trebuie să respecte convențiile numelor de domenii DNS (de exemplu nu trebuie să se termine cu un Top Level Domain). A doua deosebire constă în felul în care este găsită adresa IP asociată unei etichete. Dacă în cazul DNS se folosesc servere dedicate, în cazul aliasurilor este interogat fișierul `/etc/hosts`. În acest fișier sunt trecute corespondențele între etichete și adrese IP.

Un exemplu de fișier hosts este următorul:

```
root@asgard:/home/george# cat /etc/hosts
127.0.0.1 localhost asgard
127.0.1.1 asgard

#Servers
10.1.1.100 print.mydomain.ro my-print-server
10.1.1.101 ftp.mydomainro my-ftp-server
192.168.1.150 mail-server

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

Fișierul hosts conține mai multe linii de forma:

```
adresa_IP numele_statiei [eticheta] [eticheta] ...
```

Câmpul cel mai din stânga este adresa de IP. Urmează apoi numele principal al stației, și etichetele care se asociază cu această adresă, despărțite prin spațiu.

De obicei, atunci când sunt configurate și servere de DNS și aliasuri, aliasurile au prioritate și vor fi verificate primele.

8.4.6 Configurarea numelui stației curente

Pentru stația curentă se poate configura un nume local. Folosind oricare dintre următoarele comenzi se poate afla numele stației:

```
ubuntu@ubuntu:~$ hostname

ubuntu@ubuntu:~$ uname -n

ubuntu@ubuntu:~$ cat /proc/sys/kernel/hostname

ubuntu@ubuntu:~$ sysctl kernel.hostname
```

Numele stației se poate modifica temporar folosind comanda **hostname**, ca în exemplul următor:

```
ubuntu@ubuntu:~$ hostname
ubuntu

ubuntu@ubuntu:~$ hostname uso

ubuntu@ubuntu:~$ hostname
uso
```

Pentru a realiza o configurare permanentă se va modifica fișierul `/etc/hostname` care conține numele stației. Pentru a determina recitirea configurației din acest fișier se va rula următorul script:

```
ubuntu@ubuntu:~$ /etc/init.d/hostname.sh
```

8.5 Testarea configurațiilor de rețea

Odată ce s-au realizat configurațiile de rețea, acestea trebuie testate. Pentru testarea lor sunt folosite în mod uzual două utilitare: **ping** și **traceroute**. Aceste utilitare sunt folosite și pentru a depista problemele uzuale care pot apărea în funcționarea unei rețele.

8.5.1 Ping

Utilitarul **ping** este folosit pentru a verifica conectivitatea între două stații. El testează conexiunea fizică și parte din cea logică, folosindu-se de protocoalele IP și ICMP (*Internet Control Message Protocol*) pentru a trimite și pentru a primi mesaje.

Ping primește adresa IP a stației destinație și va trimite către aceasta un mesaj `ECHO REQUEST`. Atunci când acest mesaj este recepționat, stația destinație va răspunde cu un mesaj similar, `ECHO REPLY`. Dacă răspunsul destinației ajunge înapoi la sursă, conexiunea dintre ele funcționează. În caz contrar există probleme care împiedică cele două stații că comunice.

Principalii parametri ai comenzii ping sunt următorii:

```
ping [-f] [-c count] [-i interval] [-I interface] destination
```

Fără nicio opțiune, **ping** va trimite un număr nelimitat de mesaje către destinație, până când va fi întrerupt:

```
root@asgard:/home/george# ping 10.1.1.1
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data.
64 bytes from 10.1.1.1: icmp_seq=1 ttl=255 time=0.395 ms
64 bytes from 10.1.1.1: icmp_seq=2 ttl=255 time=0.359 ms
[CTRL+C]
--- 10.1.1.1 ping statistics ---
2 packets transmitted, 2 received, 0\% packet loss, time 8998ms
rtt min/avg/max/mdev = 0.341/0.400/0.707/0.106 ms
```

Opțiunea `-c` specifică numărul de pachete trimise, iar opțiunea `-i` specifică intervalul în secunde între două pachete trimise:

```
root@asgard:/home/george# ping 10.1.1.1 -c 3 -i 1
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data.
64 bytes from 10.1.1.1: icmp_seq=1 ttl=255 time=0.326 ms
64 bytes from 10.1.1.1: icmp_seq=2 ttl=255 time=0.459 ms
64 bytes from 10.1.1.1: icmp_seq=3 ttl=255 time=0.361 ms
--- 10.1.1.1 ping statistics ---
3 packets transmitted, 3 received, 0\% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.326/0.382/0.459/0.056 ms
```

Opțiunea `-I` specifică pe ce interfață se vor trimite pachetele:

```
root@asgard:/home/george# ping 10.1.1.1 -I eth0 -c 3
PING 10.1.1.1 (10.1.1.1) from 10.1.1.2 eth0: 56(84) bytes of data.
64 bytes from 10.1.1.1: icmp_seq=1 ttl=255 time=0.331 ms
64 bytes from 10.1.1.1: icmp_seq=2 ttl=255 time=0.326 ms
64 bytes from 10.1.1.1: icmp_seq=3 ttl=255 time=0.367 ms

--- 10.1.1.1 ping statistics ---
3 packets transmitted, 3 received, 0\% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.326/0.341/0.367/0.023 ms
```

Opțiunea `-f` permite transmiterea unui număr nelimitat de pachete `ECHO REQUEST`, fără să se aștepte răspunsul `ECHO REPLY`. În acest fel destinația este inundată (flood) de mesaje. Datorită potențialului pe care îl are această opțiune de a fi folosită în atacuri DoS (vezi secțiunea 10.4.1), parametrul `-f` poate fi folosit doar de către `root`:

```
root@asgard:/home/george# ping -f 10.1.1.1
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data.
.............................................
.............................................
CTRL+C
--- 10.1.1.1 ping statistics ---
100 packets transmitted, 3 received, 97% packet loss, time 8148ms
rtt min/avg/max/mdev = 0.240/0.389/2.291/0.450 ms, ipg/ewma 12.053/0.364
ms
```

8.5.2 Traceroute

Utilitarul **`traceroute`** este folosit pentru a testa calea pe care o urmează pachetele între sursă și destinație. El va afișa câte un mesaj pentru fiecare stație/server/echipament de rețea prin care trec mesajele.

Pentru a realiza acest lucru, traceroute se folosește de câmpul **TTL** (*Time To Live*) prezent în antetul fiecărui pachet. Din motive ce țin de felul în care funcționează o rețea este posibil ca în drumul între sursă și destinație să apară bucle. Dacă un pachet intră într-o astfel de buclă, el va fi trimis la infinit între stațiile ce aparțin buclei și nu va ajunge niciodată la destinație. Pentru a preveni o asemenea situație, în antetul fiecărui pachet a fost introdus câmpul TTL. Acest câmp conține numărul maxim de echipamente de rețea prin care poate trece un pachet înainte de a fi aruncat.

Dacă un pachet are câmpul TTL cu valoarea inițială (atunci când pleacă de la sursă) egală cu 10, atunci el poate trece prin maxim 10 echipamente de rețea. Al zecelea

echipament va arunca pachetul și va trimite un mesaj ICMP sursei prin care o va înștiința de acest lucru. Trimițând acest mesaj, al zecelea echipament își va anunța identitatea către sursa pachetului.

Pe acest sistem se bazează funcționarea utilitarului traceroute. Se trimit pachete ce au câmpul TTL configurat la o valoare din ce în ce mai mare. Atunci când valoarea TTL este 1, va răspunde primul echipament de rețea de pe cale și va spune că a aruncat pachetul trimis. Atunci când valoarea TTL este 2, va răspunde al doilea echipament de pe cale și va spune că a aruncat pachetul trimis. În felul aceasta toate echipamentele de pe parcurs vor fi identificate unul câte unul.

Parametrii uzuali ai comenzii **`traceroute`** sunt următorii:

```
traceroute [-f first_ttl] [-m max_ttl] host
```

`host` reprezintă destinația până la care se dorește aflarea rutei:

```
george@asgard:~$ traceroute rol.ro
traceroute to rol.ro (195.95.229.171), 30 hops max, 40 byte packets
 1  10.1.1.1 (10.1.1.1)  0.429 ms  0.332 ms  0.248 ms
 2  10.100.0.1 (10.100.0.1)  0.809 ms  0.657 ms  0.527 ms
 3  backbone.hertza.ro (89.165.146.177)  6.286 ms  1.765 ms  2.493 ms
 4  84.234.111.41 (84.234.111.41)  6.738 ms  7.507 ms  11.869 ms
 5  84.234.107.154 (84.234.107.154)  4.406 ms  1.603 ms  1.625 ms
 6  195.95.229.171 (195.95.229.171)  2.715 ms  1.676 ms  1.230 ms
```

Pentru fiecare echipament de pe parcurs este afișată adresa lui și timpul de răspuns pentru trei încercări succesive. În cazul în care pentru un mesaj trimis nu este primit răspuns, se va afișa caracterul `*`. Parametrul `-f` permite configurarea valorii câmpului TTL din primul mesaj trimis. Valoarea câmpului TTL din mesajele următoare va depinde de aceasta și se va incrementa cu 1 la fiecare mesaj.

```
george@asgard:~$ traceroute -f 4 rol.ro
traceroute to rol.ro (195.95.229.171), 30 hops max, 40 byte packets
 4  84.234.111.41 (84.234.111.41)  2.008 ms  8.286 ms  10.525 ms
 5  84.234.107.154 (84.234.107.154)  3.399 ms  4.614 ms  5.497 ms
 6  195.95.229.171 (195.95.229.171)  4.458 ms  2.901 ms  2.947 ms
```

Parametrul `-m` configurează valoarea maximă a câmpului TTL care va fi încercată în cazul în care destinația nu a fost atinsă. Dacă paramerul nu este specificat, se vor testa implicit căi cu lungimi de până la 30 de echipamente.

```
george@asgard:~$ traceroute -m 3 rol.ro
traceroute to rol.ro (195.95.229.171), 4 hops max, 40 byte packets
 1  10.1.1.1 (10.1.1.1)  0.404 ms  0.402 ms  0.326 ms
 2  10.100.0.1 (10.100.0.1)  0.623 ms  8.407 ms  2.069 ms
 3  backbone.hertza.ro (89.165.146.177)  9.707 ms  6.819 ms  7.895 ms
```

8.5.3 Cum se depistează problemele uzuale în cazul configurărilor de rețea

Atunci când apar probleme în configurările de rețea, pentru depistarea lor există două abordări: de sus în jos (*top-down*) și de jos în sus (*bottom-up*). Partea de sus (*top*) a unei rețele se referă la partea de aplicații, iar partea de jos (*bottom*) se referă la conexiunile fizice între echipamente.

În continuare va fi analizată abordarea de jos în sus, deoarece majoritatea problemelor dintr-o rețea sunt cauzate de conexiunile fizice între echipamente.

Să presupunem că nu poate fi accesată pagina web `www.google.ro`. În cazul acestei abordări primul lucru care trebuie verificat este faptul că există legătură fizică între stație și switch sau punctul de acces fără fir. În cazul legăturilor la Ethernet (realizate prin cablu UTP) conectivitatea se poate verifica examinând led-ul de link de pe placa de rețea. Dacă led-ul este aprins, există conexiune între stație și switch. Dacă led-ul este stins, trebuie verificat cablul de rețea sau funcționarea switch-ului.

O altă metodă folosită pentru a verifica starea link-ului pentru o interfață de rețea, utilă în special atunci când nu există acces fizic la stație, este utilizarea unui program de genul **`ethtool`**. Printre altele, acesta oferă informații detaliate despre tipul de legătură fizică existentă și starea ei. Pentru a vedea starea unei interfețe utilitarul se apelează având ca parametru numele interfeței:

```
root@asgard:/home/george# ethtool eth0
Settings for eth0:
        Supported ports: [ TP MII ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
        Advertised auto-negotiation: Yes
        Speed: 100Mb/s
        Duplex: Full
        Port: MII
        PHYAD: 32
        Transceiver: internal
        Auto-negotiation: on
        Supports Wake-on: pumbg
        Wake-on: d
        Current message level: 0x00000007 (7)
        Link detected: yes
```

În cazul în care `Link detected` are valoarea `yes`, există conexiune între stație și switch. Dacă are valoarea `no`, trebuie verificat cablul de rețea sau funcționarea switch-ului.

Următorul pas este verificarea existenței unui gateway și a conexiunii cu acesta. Pentru a verifica faptul că un gateway a fost configurat trebuie analizat fișierul `/etc/network/ interfaces` sau rezultatul comenzii **`route`**, funcție de configurație. Dacă niciun gateway nu este configurat, trebuie configurat unul. Dacă gateway-ul este configurat, trebuie verificată legătura către acesta cu ajutorul comenzii **`ping`**.

După verificarea gateway-ului, trebuie verificat DNS-ul. Pentru aceasta trebuie analizat fișierul `/etc/resolv.conf`. În cazul în care nu este configurat niciun server DNS, trebuie configurat unul. Dacă este configurat un server, trebuie testată legătura cu acesta cu ajutorul comenzii **`ping`**.

Pasul următor este verificarea conectivității cu un alt server din Internet. Se poate încerca accesarea unei alte pagini web, se poate testa conectivitatea cu un alt server cu ajutorul comenzii **`ping`** sau se poate încerca verificarea căii către `google.ro` cu

ajutorul comenzii **traceroute**. Folosirea **traceroute** oferă avantajul depistării echipamentului de rețea în care se opresc pachetele.

8.6 Studii de caz

8.6.1 Verificarea în linie de comandă a parametrilor de rețea în Windows

Cea mai ușoară metodă de a verifica parametrii de rețea în Windows este folosirea utilitarului **ipconfig**. Acesta se apelează din linie de comandă. Pentru a porni interpretorul de comenzi din Windows XP, se merge la Start→Run, unde în câmpul Open se introduce **cmd** și se apasă pe OK.

Informații despre modul de utilizare al comenzii **ipconfig**, precum și despre parametrii disponibili, se pot obține utilizând parametrul `/?`:

```
C:\Documents and Settings\George >ipconfig /?
```

Apelarea comenzii **ipconfig** fără niciun parametru va afișa parte din parametrii de rețea:

```
C:\Documents and Settings\George>ipconfig
Windows IP Configuration

Ethernet adapter Local Area Connection:
        Media State . . . . . . . . . . . : Media disconnected

Ethernet adapter Wireless Network Connection 2:

        Connection-specific DNS Suffix  . : lan
        IP Address. . . . . . . . . . . . : 192.168.1.70
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.1.254
```

Pentru a vedea toți parametrii de rețea, inclusiv serverul DNS și gateway-ul implicit, se folosește parametrul `/all`:

```
C:\Documents and Settings\George>ipconfig /all
Windows IP Configuration
        Host Name . . . . . . . . . . . . : Asgard
        Primary Dns Suffix  . . . . . . . :
        Node Type . . . . . . . . . . . . : Mixed
        IP Routing Enabled. . . . . . . . : No
        WINS Proxy Enabled. . . . . . . . : No
        DNS Suffix Search List. . . . . . : lan

Ethernet adapter Local Area Connection:
        Media State . . . : Media disconnected
        Description . . . : Broadcom 440x 10/100 Integrated Controller
        Physical Address. : 00-40-F4-CC-C5-41

Ethernet adapter Wireless Network Connection 2:
        Connection-specific DNS Suffix  . : lan
        Description . . . . : Atheros AR5006X Wireless Network Adapter
        Physical Address. . . . . . . . . : 00-19-7E-0B-7E-65
```

```
19          Dhcp Enabled. . . . . . . . . . . : Yes
20          Autoconfiguration Enabled . . . . : Yes
21          IP Address. . . . . . . . . . . . : 192.168.1.70
22          Subnet Mask . . . . . . . . . . . : 255.255.255.0
23          Default Gateway . . . . . . . . . : 192.168.1.254
24          DHCP Server . . . . . . . . . . . : 192.168.1.254
25          DNS Servers . . . . . . . . . . . : 192.168.1.254
26          Lease Obtained. . . . . . . . . . : 11 august 2007 16:17:26
27          Lease Expires . . . . . . . . . . : 12 august 2007 16:17:26
```

Configurarea parametrilor de rețea din line de comandă în Windows este mai dificil de realizat decât în Linux. Pentru a putea face acest lucru se folosește utilitarul **`netsh`**. Informații despre modul de utilizare al acestuia se pot obține folosind parametrul `/?`:

```
1 C:\Documents and Settings\George >netsh /?
```

8.6.2 Configurări de rețea în Windows Vista

Modul de configurare al unei interfețe de rețea în Windows Vista este asemănător cu cel din Windows XP. Diferența importantă între ele este dată de modul în care se accesează fereastra de configurare a parametrilor.

Din meniul Control Panel se alege *Network and Internet* (figura 8.5).

Figura 8.5: Selectarea Network and Internet din Control Panel

Apoi se selectează *Network Sharing Center* (figura 8.6).

Figura 8.6: Selectarea Network Sharing Center

Din noua fereastră se alege *Manage Network Connections* (figura 8.7).

Urmând ca apoi să fie afișată o listă cu interfețele de rețea disponibile. Prin efectuarea unui click dreapta pe simbolul asociat unei interfețe va apărea un meniu contextual asemănător cu cel din Windows XP (figura 8.8).

Prin selectarea opțiunii *Properties* din acest meniu va fi afișată fereastra de configurare a parametrilor interfeței (figura 8.9).

Pentru a configura adresa IP a interfeței, gateway-ul și serverul DNS, se alege itemul *Internet Protocol Version 4*.

Fereastra de configurare a parametrilor este identică celei din Windows XP (figura 8.10).

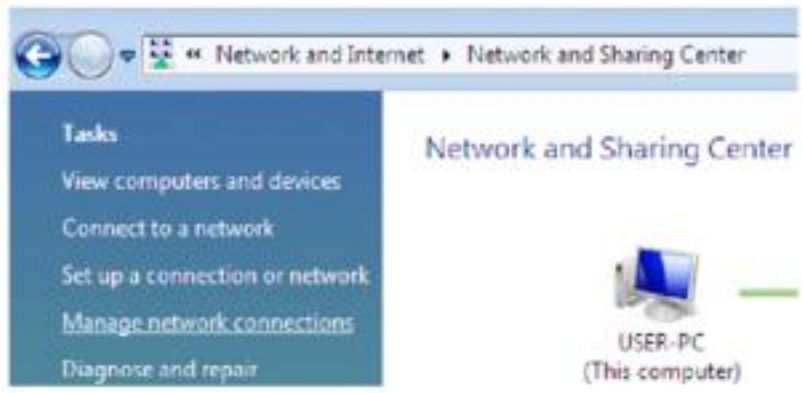


Figura 8.7: Selectarea Manage Network Connections

Figura 8.8: Meniul contextual corespunzător unei interfețe de rețea

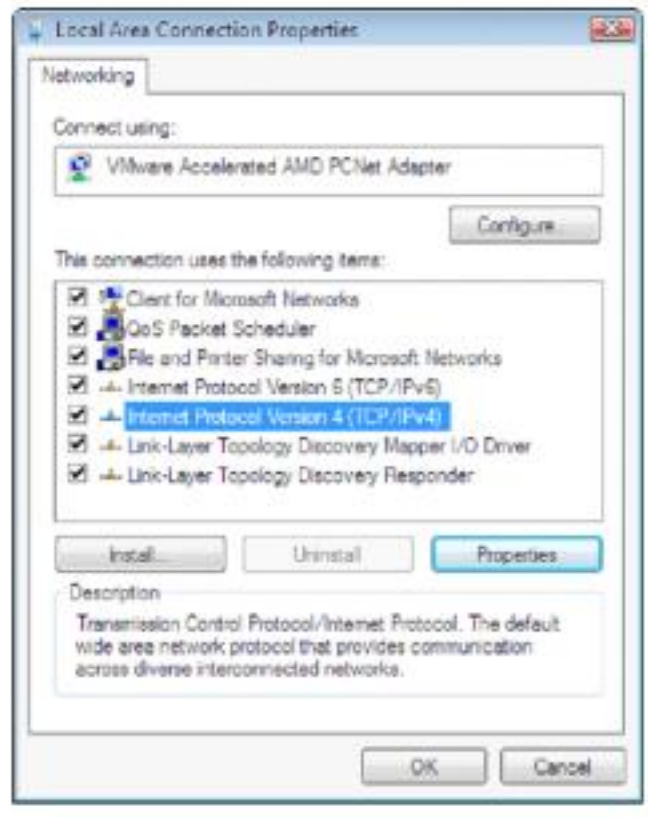


Figura 8.9: Proprietățile unei interfețe de rețea

În cazul în care este aleasă opțiunea *Obtain an IP address automatically*, adresa IP a interfeței va fi configurată prin DHCP. Pentru a specifica manual parametrii interfeței trebuie aleasă opțiunea *Use the following IP address*.

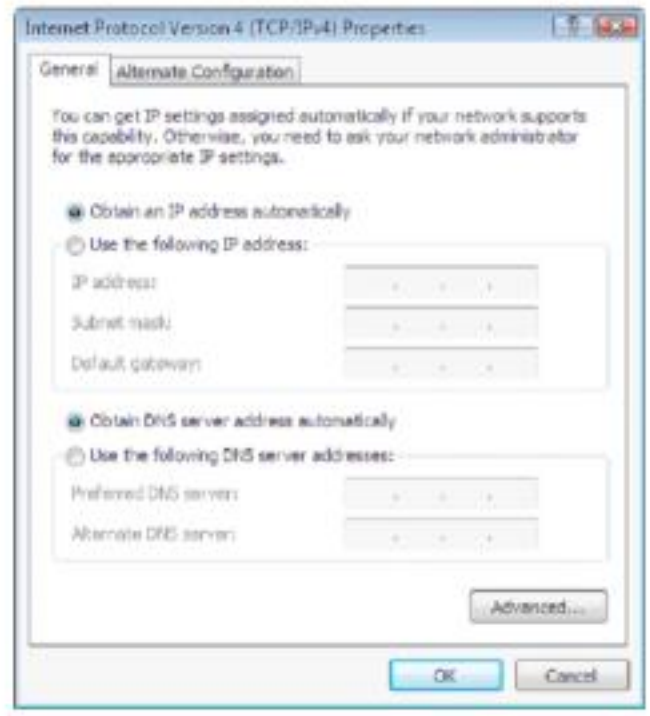


Figura 8.10: Configurarea parametrilor de rețea

8.6.3 Configurarea PPPoE în Linux

PPP (*Point to Point Protocol*) este un protocol utilizat pentru comunicația punct-la-punct (comunicația dintre două noduri conectate direct între ele). PPP permite realizarea de legături peste conexiuni fizice seriale, linii telefonice, conexiuni radio sau fibră optică.

Dintre implementările PPP, cele mai uzuale sunt cele folosite de conexiunile de tip dial-up și PPPoE.

PPPoE (*PPP over Ethernet*) este un protocol care permite încapsularea (includerea) cadrelor PPP în cadre Ethernet, cel mai frecvent pentru a putea fi trimise printr-o placă de rețea obișnuită (ce folosește cablu UTP).

PPPoE este folosit uzual în cadrul serviciilor de tip ADSL sau Internet prin cablu TV. Dacă legătura fizică dintre calculator și modem se realizează prin cablu UTP, atunci este foarte probabil ca legătura logică dintre calculator și modem să folosească PPPoE. Pentru a putea să acceseze Internetul, utilizatorul trebuie mai întâi să stabilească o conexiune PPP cu modemul.

Stabilirea unei conexiuni PPP necesită, de cele mai mule ori, autentificarea utilizatorului. Aceasta se realizează pe baza unui nume de utilizator și a unei parole furnizate de către ISP (*Internet Service Provider* – furnizorul de servicii de Internet).

Pentru configurarea parametrilor conexiunii PPP, inclusiv a parametrilor necesari autentificării, cea mai ușoară metodă este folosirea utilitarului **`pppoeconf`**. Instalarea acestuia depinde de instalarea tuturor pachetelor necesare funcționării protocolului PPP, astfel încât, pe o distribuție ce folosește pachete cum este Ubuntu sau Debian, odată cu **`pppoeconf`** se vor instala și toate dependințele sale.

Pe de altă parte, odată configurată conexiunea PPP cu ajutorul pppoeconf, acesta se va ocupa de realizarea tuturor fișierelor de configurare, inclusiv de adăugarea unui script care la pornirea sistemului de operare va realiza conectarea prin PPP.

Pentru configurarea PPPoE sunt necesare următoarele fișiere (ele sunt create sau actualizate automat de către **`pppoeconf`**):

- `/etc/network/interfaces`
- `/etc/ppp/pap-secrets`
- `/etc/ppp/peers/dslprovider`

Ultimul dintre fișiere poate avea orice nume și se poate afla în orice loc, însă trebuie specificat în cadrul fișierului `/etc/network/interfaces`.

Un exemplu de `/etc/network/interfaces` este următorul:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
iface eth0 inet manual

auto dsl-provider
iface dsl-provider inet ppp
pre-up /sbin/ifconfig eth0 up # line maintained by pppoeconf
provider dsl-provider

auto eth0
```

Autentificarea conexiunii PPP se realizează folosind protocoale specializate. Un exemplu de fișier `/etc/ppp/pap-secrets`, în care sunt trecute informațiile necesare autentificării, este următorul:

```
[...]

# INBOUND connections

# Every regular user can use PPP and has to use passwords from
/etc/passwd
* hostname  ""  *

# UserIDs that cannot use PPP at all. Check your /etc/passwd and add any
# other accounts that should not be able to use pppd!
guest hostname  "*" -
master  hostname  "*" -
root    hostname  "*" -
support hostname  "*" -
stats hostname  "*" -

# OUTBOUND connections

# Here you should add your userid password to connect to your providers
via
# PAP. The * means that the password is to be used for ANY host you
connect
# to. Thus you do not have to worry about the foreign machine name. Just
# replace password with your password.
# If you have different providers with different passwords then you
better
# remove the following line.
# * password

"student" * "usorules"
```

În fișierul `/etc/ppp/peers/dslprovider` sunt trecute informații legate de parametrii conexiunii PPP, alții decât cei de autentificare:

```
# Minimalistic default options file for DSL/PPPoE connections

noipdefault
defaultroute
replacedefaultroute
hide-password
noauth
persist
#mtu 1492
#persist
#holdoff 20
plugin rp-pppoe.so eth0
user "student"
usepeerdns
```

Pentru a activa o conexiune PPP se folosește comanda **`pon`**. Ea este echivalentă cu parametrul up al comenzii **`ifconfig`**. Pentru a dezactiva o conexiune PPP se folosește comanda **`poff`**, echivalentă cu parametrul `down` al comenzii **`ifconfig`**.

Un amănunt important de precizat este că, pentru a funcționa prin PPPoE, o interfață de rețea Ethernet nu necesită configurarea unei adrese IP. Interfața de rețea Ethernet este folosită indirect pentru conectarea la Internet: prin ea sunt trimise datele către modemm acesta asigurând mai departe legătura către Internet.

În acest caz adresa IP este asociată cu o interfața virtuală numită `ppp0`:

```
root@asgard:/home/george# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet6 addr: fe80::240:f4ff:febb:1701/64 Scope:Link
          [...]

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]

ppp0      Link encap:Point-to-Point Protocol
          inet addr:192.168.10.3  P-t-P:10.0.0.1  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1492  Metric:1
          RX packets:176 errors:0 dropped:0 overruns:0 frame:0
          TX packets:180 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:105255 (102.7 KiB)  TX bytes:13991 (13.6 KiB)
```

După cum se poate observa, interfața `eth0` nu are configurată adresa IP. În schimb interfața `ppp0`, asociată conexiunii PPP, este cea care fost configurată cu adresa IP `192.168.10.3`.

Cuvinte cheie

- rețea de calculatoare
- LAN, MAN, WAN
- protocolul IP; adresă IP
- adresă MAC
- Ethernet
- UTP

- ruter
- unicast, multicast, broadcast
- adresă de (sub)rețea
- default gateway
- DNS
- switch
- Access Point
- DHCP
- configurare temporară/permanentă;
- configurare statică/dinamică
- ifconfig
- /etc/network/interfaces
- /etc/resolv.conf
- /etc/hosts
- route
- DHCP
- dhclient
- ping
- traceroute
- ipconfig
- PPPoE

Întrebări

1. Un utilizator încearcă să testeze conectivitatea către `cs.pub.ro` cu ajutorul comenzii **`ping`**: "ping cs.pub.ro" și testul reușește. Totuși, dacă utilizatorul tastează în bara de adrese a browser-ului "cs.pub.ro", pagina web respectivă nu poate fi accesată. Care poate fi motivul pentru care cele două teste au rezultate diferite?
   - ❑ serverul web de la adresa `cs.pub.ro` este oprit
   - ❑ utilizarea comenzii **`ping`** este greșită
   - ❑ comanda **`ping`** a oprit serverul web
   - ❑ utilitarul **`ping`** nu oferă informații despre conexiunea cu `cs.pub.ro`
2. Se poate configura adresa 10.138.257.134 pe interfața de rețea eth0?
   - ❑ da, oricând
   - ❑ da, doar dacă este singura interfața a sistemului
   - ❑ da, doar în situația în care este configurat un server DNS
   - ❑ nu
3. Dându-se o stație care are configurată adresa IP `172.16.150.200` și masca de rețea `255.255.255.0`, care este adresa rețelei din care face parte stația?
   - ❑ `172.0.0.0`
   - ❑ `172.16.0.0`
   - ❑ `172.16.150.0`
   - ❑ `172.16.150.200`
4. Deși legătura la Internet funcționa, utilizatorul Dorel a dorit să testeze conectivitatea cu gateway-ul cu ajutorul comenzii **`ping -f 192.168.10.1`**.

La scurt timp el a fost contactat de administratorul de rețea care l-a rugat să își verifice calculatorul împotriva virușilor. Acest lucru s-a petrecut deoarece:

- ❑ Dorel foloseste Linux și nu Windows, la fel ca majoritatea celorlalți utilizatori.
- ❑ Dorel nu a repornit calculatorul de mult timp.
- ❑ Dorel a generat foarte mult trafic în timp scurt, lucru specific virușilor.
- ❑ Dorel nu obisnuiește să cripteze email-urile trimise.

5. Rolul serviciului DNS este de a:
   - ❑ asocia nume de domenii cu adrese MAC
   - ❑ asocia nume de domenii cu adrese IP
   - ❑ asocia adrese IP cu adrese MAC
   - ❑ asocia adresa IP cu numele utilizatorului
6. În fișierul `/etc/network/interfaces` se găsește următoarea linie:

```
1 iface eth0 inet dhcp
```

   În linia de comandă este dată următoarea comandă:

```
1 ifconfig eth0 192.168.124.150
```

   după care stația este repornită. Care va fi adresa IP a interfeței `eth0` imediat după repornirea stației?
   - ❑ `192.168.124.150`, pentru că a fost ultima adresă configurată de `root`.
   - ❑ `192.168.124.150` dacă aceasta este adresa furnizată stației de serverul DHCP
   - ❑ interfața nu va avea nicio adresă IP pentru că s-a creat un conflict prin configurarea adresei IP în două moduri
   - ❑ interfața nu va avea nicio adresă IP pentru că nu a fost dat parametrul `netmask` comenzii **`ifconfig`**
7. Configurările permanente se realizează cu ajutorul fișierului `/etc/network/ip`. Configurările temporare își pierd efectul o dată cu repornirea sistemului.
   - ❑ adevărat, fals
   - ❑ adevărat, adevărat
   - ❑ fals, fals
   - ❑ fals, adevărat
8. Masca de rețea este utilizată pentru a identifica adresa rețelei din cadrul unei adrese IP. Pentru a putea accesa serverele din Internet este necesară configurarea unui server DNS.
   - ❑ adevărat, fals
   - ❑ adevărat, adevărat

- ❑ fals, fals
- ❑ fals, adevărat

9. Cu ajutorul comenzii **`traceroute`** se poate testa:
   - ❑ Lățimea de bandă a unei conexiuni
   - ❑ Calea dintre sursă și destinație
   - ❑ Funcționarea serverului web de pe stația destinație
   - ❑ Existența unui ruter între sursă și destinație
10. Dacă un server DNS nu este specificat, care dintre următoarele comenzi este probabil să funcționeze:
    - ❑ **`ping www.google.ro`**
    - ❑ **`ping cs.pub.ro`**
    - ❑ **`ping localhost`**
    - ❑ **`ping dns.localdomain.ro`**

Capitolul 9

Servicii de rețea

When I took office, only high energy physicists had ever heard of what is called the World Wide Web.... Now even my cat has its own page.

Bill Clinton, 1996

Ce se învață din acest capitol?

- Conceptul de protocol și stivă de protocoale
- Modelul client-server
- Folosirea porturilor
- Execuția comenzilor la distanță: telnet, ssh, wget, curl
- Funcționarea serviciului de poștă electronică (email)
- Funcționarea serviciului de web
- Modalități de partajare a fișierelor: SMB, FTP

9.1 Concepte specifice aplicațiilor de rețea

9.1.1 Stiva TCP/IP

Sistemele de calcul au început să fie dezvoltate odată cu sfârșitul celui de al doilea război mondial. Cu toate acestea, problematica asigurării comunicației între sisteme aflate în locații la mare distanță a fost formulată abia spre începutul anilor '70, de către US DoD (*US Department of Defense* – Ministerul Apărării Naționale al Statelor Unite). Proiectul US DoD a creat și un cadru de standardizare pentru soluțiile de comunicație, denumit stiva de protocoale TCP/IP (*Transmission Control Protocol/Internet Protocol*).

Un **protocol** este un set de reguli care guvernează modul în care două dispozitive schimbă informație într-o rețea și asigură coerența comunicației.

O **stivă de protocoale** reprezintă un set de protocoale ce comunică între ele prin funcții (numite primitive) clar definite.

O stivă de protocoale este în general privită ca o succesiune de niveluri. Fiecare dintre niveluri este descris de:

- un set de funcții pe care le îndeplinește;
- un set de parametri pe care îi așteaptă de la nivelul inferior;
- serviciile pe care le oferă nivelului imediat superior.

Protocoalele dintr-o stivă nu vor putea comunica între ele direct decât dacă se află pe niveluri adiacente.

Descrierea unei soluții de comunicare prin prisma unei stive de protocoale oferă în primul rând avantajul flexibilității procesului de standardizare. Astfel, schimbarea (sau simpla optimizare) unui protocol din stivă poate fi făcută fără a solicita modificări în restul protocoalelor.

Începuturile rețelelor de date au fost marcate de existența mai multor protocoale de comunicație, dezvoltate în paralel, de vendori diferiți. Problema consta în faptul că, deși două protocoale, de la doi vendori diferiți, descriau amândouă foarte bine specificațiile propriilor dispozitive, nu puteau să comunice unul cu celălalt. Astfel, pentru a face posibilă intercomunicarea între echipamentele de rețea produse de vendori diferiți, s-a introdus conceptul de standardizare a unui protocol. Odată ce un protocol este acceptat ca standard, el devine dintr-un set de reguli, un **set comun de reguli**, fiind recunoscut din punct de vedere procedural și funcțional de toți vendorii.

Pentru a descrie prelucrările efectuate asupra datelor trimise de la utilizator spre mediul de transmisie, stiva TCP/IP identifică patru niveluri, prezentate în figura 9.1: `Aplicație`, `Transport`, `Rețea` (sau `Internet`) și `Acces la mediu`.

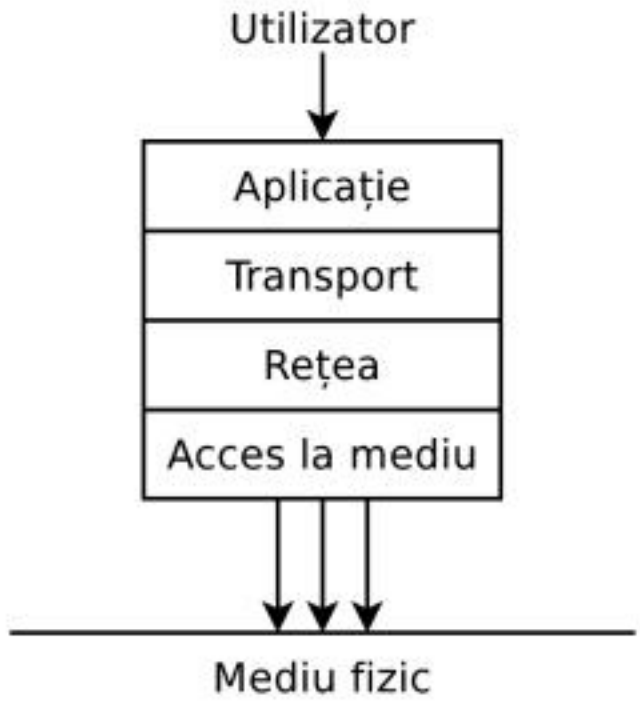


Figura 9.1: Stiva TCP/IP

Nivelul aplicație

Nivelul aplicație este cel mai apropiat de utilizator și oferă acestuia modalitatea de accesare a serviciului de rețea. Un serviciu de rețea poate fi implementat prin unul sau mai multe protocoale de nivel aplicație.

Protocoalele de la acest nivel vor conține reguli referitoare la interacțiunea cu utilizatorul, formatarea și reprezentarea datelor, precum și la controlul dialogului între componentele serviciului.

În funcție de serviciul oferit, protocoalele de nivel aplicație pot fi grupate în:

- serviciul de transfer al fișierelor, ce poate fi implementat folosind FTP (*File Transfer Protocol*), HTTP (*HyperText Transfer Protocol*) etc.
- serviciul de poștă electronică, ce se bazează pe următoarele protocoale: SMTP (*Simple Mail Transfer Protocol*), POP3 (*Post Office Protocol 3*), IMAP (*Internet Mail Access Protocol*)
- serviciul de executarea comenzilor la distanță: **`telnet`**, **`ssh`**, **`rlogin`**
- alte protocoale: DNS (*Domain Name System*), DHCP (*Dynamic Host Configuration Protocol*), SNMP (*Simple Network Management Protocol*)

Nivelul transport

Nivelul transport este responsabil cu probleme legate de siguranță, control al fluxului și corecție de erori. Tot la nivelul transport este realizată multiplexarea comunicației prin specificarea **porturilor sursă și destinație**. Cele mai importante protocoale de nivel transport sunt TCP (*Transmission Control Protocol*) și UDP (*User Datagram Protocol*).

TCP este un **protocol orientat conexiune**, adică înainte de a trimite datele va negocia parametrii de comunicație între entitățile participante la comunicație, iar apoi pe tot parcursul va asigura reordonarea pachetelor ajunse în ordine greșită la destinație, sau retransmisia celor pierdute.

Pentru a înțelege mai bine noțiunea de protocol orientat pe conexiune, se poate face o analogie cu transmiterea datelor prin rețeaua telefonică. În momentul efectuării unei convorbiri telefonice, se stabilește între sursa și destinatie o conexiune dedicată prin rețea, înainte de începerea comunicării. Deasemenea acest exemplu relevă faptul că într-un protocol orientat pe conexiune, destinația este mereu contactată înainte de începerea transmisiei propriu-zise (în analogia cu soneria telefonului).

UDP este un **protocol neorientat conexiune**. Astfel, UDP nu se va preocupa de inițierea sau terminarea conexiunii, de pierderea pachetelor sau doar a succesiunii acestora. UDP va începe transmisia fără a verifica măcar dacă destinația finală va accepta respectivul trafic.

Deși 95% din traficul Internet folosește protocolul TCP, pentru rețelele locale numeroase servicii se bazează pe UDP: DNS, SNMP, TFTP. Motivul principal îl reprezintă gradul redus al erorilor în rețelele locale (spre deosebire de WAN). UDP este mai simplu de implementat, dar și mai eficient decât TCP: antetul (informația suplimentară adăugată de fiecare protocol) UDP este doar de 8 octeți, față de 20 de octeți cât are TCP.

Nivelul rețea

Nivelul rețea (Internet) se ocupă cu transmiterea pachetelor de la o sursă la o destinație precizată, indiferent în ce rețea se află aceasta. Alegerea căii pachetului prin rețelele intermediare, comutarea lui dintr-o o rețea în alta și optimizarea traseelor alese sunt responsabilitatea protocolului de rețea.

Nivelul acces la mediu

Nivelul acces la mediu se ocupă cu toate problemele legate de transmiterea efectivă a unui pachet pe o legătură fizică, incluzând și aspectele legate de tehnologii și de medii de transmisie.

9.1.2 Implementarea parțială a stivei TCP/IP

Dispozitivele de interconectare asigură transferul cât mai rapid al datelor în general pe baza unor algoritmi de decizie. Un dispozitiv de interconectare poate în același timp asigura conversia între două tipuri de rețea (două protocoale) diferite.

Criteriul principal de evaluare a eficienței unui dispozitiv de interconectare este **numărul de pachete comutate pe secundă**. În funcție de complexitatea deciziei de comutare numărul de pachete prelucrate scade. Pentru echipamentele de rețea ce folosesc un număr redus de criterii pentru decizie, se renunță la implementarea nivelurilor superioare din stiva de protocoale TCP/IP.

O primă clasă de dispozitive de interconectare își bazează deciziile de comutare doar pe informațiile protocoalelor de nivel acces la mediu. În această categorie intră **switch-urile**, dar și **AP**-urile (*access point*).

A doua clasă de dispozitive va analiza atât informațiile de nivel acces la mediu, cât și pe cele de nivel rețea, procesul de decizie fiind mai complex, dar și mai lent decât în primul caz. Cel mai important dispozitiv de interconectare de nivel rețea este **ruterul**.

Dispozitivele de interconectare pot implementa doar o parte a stivei de protocoale. Spre exemplu un ruter nu trebuie să înțeleagă conținutul unui pachet de date pentru a putea lua decizia de trimitere pe calea optimă. Astfel, ruterul are nevoie doar de informațiile de nivel acces la mediu și nivel rețea și va implementa doar primele două niveluri ale stivei TCP/IP.

Figura 9.2: Comunicația între două stații aflate în rețele diferite

Considerând o topologie simplă, prezetată în figura 9.2, în care un client de web aflat pe stația B acceseaza un server aflat pe stația A. Cele două stații se află în rețele locale diferite conectate printr-un ruter. Rețeua din stânga este o rețea Ethernet, în vreme ce în rețeua din dreapta AP-ul (acces point) este conectat prin Ethernet cu ruterul R și oferă conectivitate fară fir pentru stația B.

Modelul de comunicație poate fi descris generic folosind stiva de protocoale TCP/IP precum în figura 9.3.

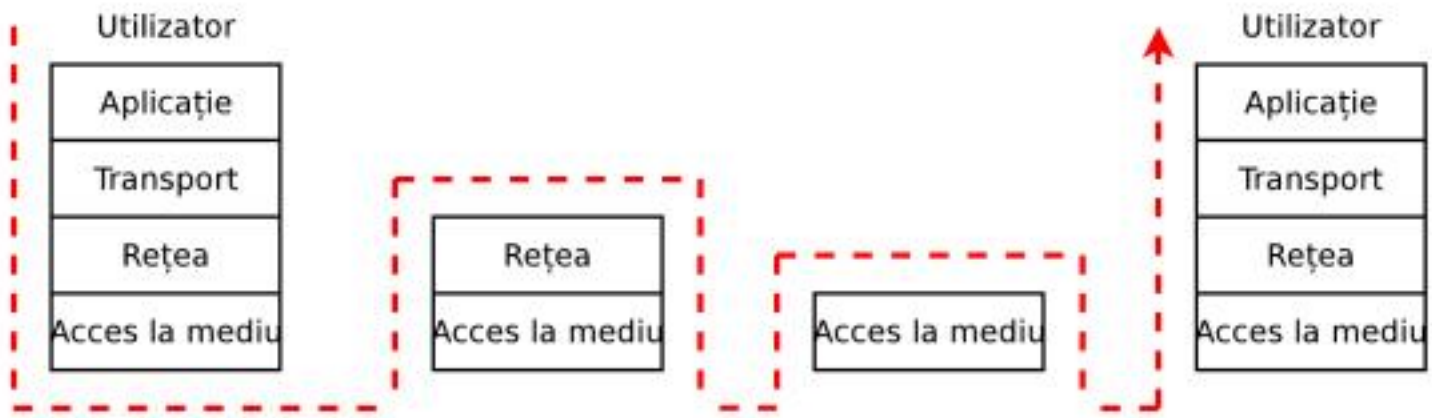


Figura 9.3: Descrierea generică a comunicației

Pentru topologia considerată în figura 9.2, la nivelul aplicație se folosește serviciul de web bazat pe protocolul HTTP. La nivelul transport și nivelul rețea se folosește TCP, respectiv IP. Spre deosebire de nivelul aplicație, unde putem avea o largă diversitate de protocoale folosite, la nivelurile transport și rețea peste 95% din trafic-ul în Internet este trafic TCP/IP.

Pentru nivelul acces la mediu în rețelele locale există două tehnologii dominante: Ethernet (IEEE 802.3) și WLAN (*Wireless* LAN, IEEE 802.11). Conectivitatea între cele două standarde poate fi realizată fie de ruter (cu o interfață 802.11 și una Ethernet), fie de un AP (*acces point*).

Un AP nu are nevoie să interpreteze informația aflată la nivelul rețea pentru a realiza comutarea între rețeaua Ethernet și rețeaua fără fir.

Cele două dispozitive de interconectare (ruterul și AP-ul) au fost figurate ca având câte o stiva parțială de protocoale pe fiecare dintre interfețele de rețea. Stiva interfeței de intrare va fi folosită pentru a lua decizia de comutare a pachetului (alegerea interfeței de ieșire), un echipament de interconectare putând avea mai mult de două interfețe. După determinarea interfeței de ieșire se va folosi stiva acesteia (parametrii specifici acestei interfețe) pentru eventuala reformatare a pachetului.

Pentru exemplul considerat, fluxul de date între stația A și stația B, va solicita funcții din mai multe protocoale, acestea fiind prezentate în figura 9.4.

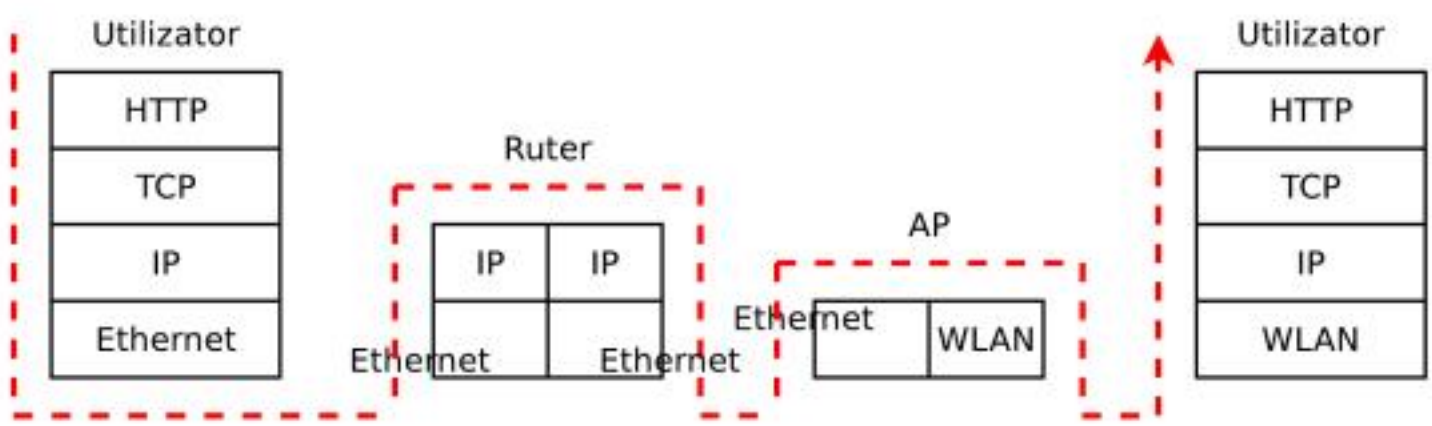


Figura 9.4: Descrierea comunicației între două stații aflate în rețele diferite

9.1.3 Modelul client-server

Pentru implementarea accesului la serviciile de rețea, arhitectura folosită este bazată pe modelul client-server. Acesta se bazează pe definirea a două componente: cea de server și cea de client.

> Un **server** este un program (sau prin extindere o stație) ce permite primirea de cereri de la alte entități din rețea, oferind acestora un serviciu.

> Un **client** este un program ce oferă mecanismul de interogare a unui server și eventual de formatare a răspunsului primit de la acesta.

Un exemplu de **model client server** este serviciul de web. Acest serviciu presupune rularea unui server HTTP (ex: Apache, Microsoft IIS etc). Aplicația client poate fi orice navigator (*broswer*).

Chiar pentru exemplul serviciului de web se poate observa că aplicația client integrează în realitate mai multe aplicații client: pentru HTTP, dar și pentru FTP, TFTP.

Modelul client-server oferă centralizarea serviciilor și a fost în primul rând promovat ca o metodă de reducere a costurilor totale a unui rețele. Ca urmare a scăderii costului resurselor hardware, principală motivație a centralizării serviciilor a devenit monitorizarea și securizarea rețelei.

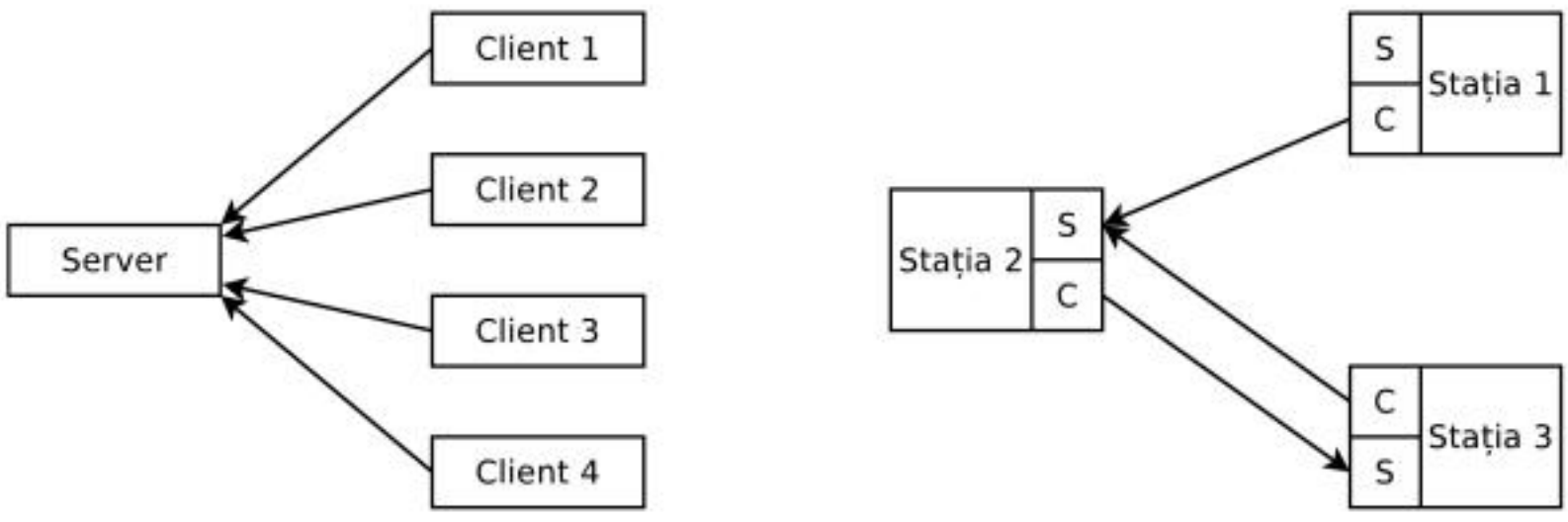


Figura 9.5: Modelul client-server și modelul peer-to-peer

Noțiunea de server s-a extins și pentru sistemele de calcul, specificațiile hardware ale unui server depinzând de serviciul pe care îl îndeplinește. Spre exemplu, un server de stocare va avea un procesor relativ modest, dar un spațiu impresionant de stocare, precum și mecanisme de redundanță a discurilor.

Modelul punct-la-punct (*peer to peer*) este de fapt un model client-server ascuns, așa cum se poate vedea și din figura 9.5. Fiecare entitate ce participă la comunicație poate juca atât rol de client cât și rol de server.

Spre deosebire de modelul client-server, modelul punct-la-punct este un model descentralizat.

Multe din programele de partajare de fișiere utilizează acest model pentru a nu folosi un server care să intermedieze comunicația. Un model client-server de partajare a fișierelor

între clienți presupune ca orice pachet schimbat între cei doi clienți să treacă prin server, ceea ce aduce atât o latență, cât și o încărcare a rețelei și a serverului.

Principalul dezavantaj al centralizării serviciilor este faptul că serverul va fi un punct critic în rețea: în momentul în care serverul nu funcționează, serviciul este întrerupt. Într-o rețea punct-la-punct acest dezavantaj este evitat.

9.1.4 Porturi

În scopul reducerii costurilor unei rețele, mai multe servicii pot fi oferite de aceeași mașină. Spre exemplu, pentru o rețea locală, serverele de DHCP, și DNS pot fi integrate pe o singură stație, deoarece resursele solicitate de fiecare serviciu sunt relativ reduse.

Cum va putea totuși un astfel de server să diferențieze între cererile de rezolvare de nume (cereri DNS) și cele de alocare de adrese IP (cereri DHCP)?

Ambele protocoale folosesc la nivelul transport UDP, deci nu vom putea lua decizia pe baza tipul protocolului de nivel transport.

Atât UDP, cât și TCP folosesc un mecanism de identificare a serviciilor (a protocolului de nivel aplicație) denumit port.

> **Portul** este un număr reprezentat pe 16 biți, ce identifică la nivelul transport (în informația adăugată de protocoalele TCP sau UDP) un serviciu specific.

Mecanismul de porturi permite atât oferirea simultană a mai multor servicii, cât și diferențierea între cereri de răspuns venite din conexiuni diferite, adică folosirea mai multor servicii simultan. Astfel, în antetul de nivel transport vom preciza atât un port destinație, ce identifică serviciul apelat, cât și un port sursă, pe care este așteptat răspunsul. O conexiune va fi descrisă prin precizarea parametrilor de nivel rețea și de nivel transport: adresa IP destinație, adresa IP sursă, port destinație și port sursă.

Dacă de pe aceeași stație sunt lansate două sesiuni de SSH către același server, se creează două conexiuni diferite având aceleași adrese IP (sursă și destinație), același port destinație, dar portul sursă diferit.

Porturile sunt numere reprezentate pe 16 biți, valorile definite pentru porturi fiind cuprinse în intervalul [1, 65535]. Porturile până la 1023 sunt rezervate pentru cele mai importante protocoale ale Internet-ului, fiind denumite porturi rezervate (*known ports*). Porturile rezervate sunt asociate unui protocol pe care serverul îl folosește în oferirea unui serviciu.

Rezumând: fiecare serviciu de rețea este oferit folosind un protocol de nivel aplicație, fiecare protocol are la rândul său asociat un port, în unele cazuri două sau mai multe.

În tabelul 9.1 sunt prezentate câteva dintre protocoalele importante și porturile folosite. Putem observa spre exemplu că serviciul de FTP are rezervate două porturi: 21 (pentru stabilirea conexiunii) și 20 (pentru transferul efectiv de fișiere).

Pentru a determina serviciile oferite pe stația locală va trebui să afișăm lista porturilor deschise (aflate în stare `LISTEN`). Pentru aceasta putem folosi utilitarul **`netsat`** (vezi secțiunea 10.3.1):

Tabelul 9.1: Asocieri port-serviciu pentru câteva servicii răspândite

Port	Serviciu
20	FTP (data)
80	HTTP
21	FTP (conexiune)
110	POP3
22	SSH
143	IMAP
23	telnet
161	SNMP
25	SMTP
443	HTTPS
53	DNS
993	IMAPS

```
root@kiwi~# netstat --tcp --listening
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:sunrpc                *:*                     LISTEN
tcp        0      0 *:www                   *:*                     LISTEN
tcp        0      0 *:webcache              *:*                     LISTEN
tcp        0      0 *:auth                  *:*                     LISTEN
tcp        0      0 *:ftp                   *:*                     LISTEN
tcp        0      0 *:smtp                  *:*                     LISTEN
```

Din rezultatul comenzii de mai sus se poate deduce că portul ftp (adică portul 21) este deschis. Altfel spus, există o aplicație ce folosește acest port pentru a accepta conexiuni. În general, serviciile sunt rulate pe porturile rezervate, dar prin configurarea serviciilor putem schimba portul pe care se vor accepta conexiunile. Pentru a obține mai multe informații despre aplicația ce rulează pe portul 21 putem folosi clientul de **`telnet`** ce va deschide o conexiune pe acest port:

```
razvan@kiwi:~$ telnet kiwi.cs.pub.ro 21
Trying 141.85.99.5...
Connected to kiwi.cs.pub.ro.
Escape character is '^]'.
220 Welcome to Kiwi FTP service. vsFTPd 2.0.3 is in service
```

Dacă oprim serviciul de FTP și reconfigurăm serverul de SSH să primească cereri pe portul 21, rezultatul comenzii **`telnet`** va fi:

```
razvan@kiwi:~$ telnet kiwi.cs.pub.ro 21
Trying 141.85.99.5...
Connected to kiwi.cs.pub.ro.
Escape character is '^]'.
SSH-2.0-OpenSSH_3.8.1p1 Debian-8.sarge.4
```

Din motive de securitate serverele pot fi configurate să nu afișeze informații referitoare la versiunea serviciului, sau să afișeze informații false.

În cazul în care se încearcă inițierea unei sesiuni de telnet pe un port ce nu este deschis va fi întors mesajul de conexiune refuzată:

```
razvan@kiwi:~$ telnet kiwi.cs.pub.ro 29
Trying 141.85.99.5...
telnet: Unable to connect to remote host: Connection refused
```

9.2 Execuția comenzilor la distanță

Serviciul de execuție al comenzilor la distanță (*remote connection*) este oferit de mai multe protocoale, cele mai importante fiind: telnet, rlogin, RSH, SSH.

Protocoalele telnet, rlogin și RSH se bazează pe realizarea unei conexiuni necriptate (în text clar). Spre deosebire de acestea, protocolul SSH stabilește o conexiune criptată, conexiune folosită atât pentru lansarea comenzilor la distanță, cât și pentru transferul de fișiere.

9.2.1 Telnet

În practică, graniță dintre serviciu de rețea și protocol de nivel aplicație este, deseori, una greu de evidențiat. Astfel, putem spune că serviciul de execuție al comenzilor la distanță este oferit de protocolul telnet, dar la fel de corect se poate vorbi de serviciul telnet. Prin serviciul telnet se înțelege posibilitatea execuției comenzilor la distanță peste o conexiune nesigură (necriptată).

Pe lângă denumirea unui protocol de nivel aplicație, și a unui serviciu de rețea, **`telnet`** este și denumirea aplicației ce implementează protocolul (a procesului care din punctul de vedere al sistemului de operare oferă serviciul de rețea).

Aplicația **`telnet`** este o aplicație client-server: procesul server se numește `telenetd`, iar clientul se numește **`telnet`**. Serverul de telnet este responsabil cu acceptarea sesiunilor inițiate de clienți. O sesiune de telnet pune la dispoziția aplicației client (după autentificare) un interpretor de comenzi ce rulează pe server. În mod implicit, nu există diferențe între resursele disponibile unui utilizator în urma rulării unui interpretor de comenzi local, și resursele accesibile printr-o sesiune de telnet de către același utilizator.

```
rookie@apple~$ telnet 141.85.99.5
Trying 141.85.99.5...
Connected to 141.85.99.5.
Escape character is '^]'.
Fedora release 7 (Moonshine)
Kernel 2.6.21-1.3194.fc7 on an i686
login: manager
Password:
manager@kiwi~$
```

În exemplul de mai sus utilizatorul `rookie` inițiază o sesiune telnet către serverul cu adresa 141.85.99.5. După afișarea mesajelor de întâmpinare (denumite MotD – *Message of the Day*) este afișat prompt-ul de login. Este folosit numele de utilizator manager și parola asociată acestui cont. După autentificare va apărea prompt-ul interpretorului de comenzi.

Protocolul telnet a fost standardizat pentru a permite clientului de telnet inițierea de conexiuni la distanță și către alte servere. Majoritatea implementărilor serverelor permit inițierea unei conexiuni prin clientul de telnet (cu unele excepții notabile, precum clienții RDS, cărora le este blocat portul 25 al serviciului SMTP).

Pentru a iniția o conexiune pe serverul de HTTP ce rulează pe stația cu adresa IP 141.85.99.5 trebuie rulată aplicația client **`telnet`** cu precizarea destinației și a portului pe care rulează serviciu (implicit 80). După stabilirea sesiunii se pot executa inclusiv comenzi ce respectă sintaxa protocolului HTTP, spre exemplu pentru a obține o pagină se folosește comanda HTTP GET, urmată de pagina ce se dorește afișată.

```
1  razvan@kiwi:~$ telnet 141.85.99.5 80
2  Trying 127.0.0.1...
3  Connected to localhost.localdomain.
4  Escape character is '^]'.
5  GET /~razvan/test.html HTTP/1.0
6
7  HTTP/1.1 200 OK
8  Date: Fri, 03 Nov 2008 19:57:10 GMT
9  Server: Apache/1.3.33 (Debian GNU/Linux) PHP/4.3.10-16
10 Last-Modified: Fri, 03 Nov 2008 19:56:40 GMT
11 ETag: "253095-44-454b9ef8"
12 Accept-Ranges: bytes
13 Content-Length: 68
14 Connection: close
15 Content-Type: text/html; charset=iso-8859-1
16
17 <br />Pagina de test pentru cursul USO - Servicii de Retea <br />
18 Connection closed by foreign host.
```

Este important de reținut că pentru terminarea unei sesiuni de telnet se poate folosi combinația `CRTL+]`.

9.2.2 SSH

SSH (*Secure Shell*) este un protocol de nivel aplicație, folosit în rețele de calculatoare pentru asigurarea unui canal securizat de transmitere a informației.

Protocolul funcționează după modelul client-server. Pentru a iniția o conexiune, clientul de SSH trimite o cerere către serverul de SSH, iar acesta din urmă, după autentificarea clientului, permite transferul de date.

Cea mai întalnită implementare a protocolului SSH 2.0 este varianta open-source numită OpenSSH.

Protocolul este adesea folosit pentru executarea comenzilor la distanță, dar poate fi deasemenea utilizat și pentru transfer securizat de fișiere sau tunelarea altor protocoale mai nesigure (FTP). Există multe alte protocoale de nivel aplicație, mai vechi, cu care se pot obține aceleași efecte precum cu SSH: FTP, rlogin, rsh, telnet. Toate aceste protocoale au însă în comun aceeași slăbiciune: folosesc metode de autentificare slabe, ce presupun trimiterea parolelor pe rețea în format text clar (*clear text*). Formatul text clar reprezintă un risc maxim de securitate, capturarea parolelor făcându-se relativ simplu de o persoană cu intenții malițioase.

Pentru asigurarea securității, SSH folosește algoritmul RSA de generare de **chei publice și private**. După cum denotă și numele celor două chei, cheia privată trebuie păstrată întotdeauna secretă, iar cea publică poate fi cunoscută de oricine. Există o strânsă relație matematică între cele două chei, care face imposibilă derivarea cheii private din cea publică, dar care permite decriptarea unui mesaj criptat cu una dintre chei, folosind cealaltă cheie. Dacă se dorește trimiterea unui mesaj privat către destinatarul X, tot ce trebuie făcut este criptarea mesajului cu cheia publică a destinatarului (fiind publică, toată lumea are acces la această cheie). Odată criptat, acest mesaj poate fi decriptat doar de destinatarul X, căci el este singurul care deține cheia privată corespunzătoare cheii publice cu care s-a făcut criptarea.

Utilizarea SSH pentru efectuarea conectării la distanță

Comanda de conectare la un server SSH are doi parametri importanți: numele utilizatorului și adresa (numele) serverului destinație. Serverul de SSH la care se va face conectarea (`securessh.pub.ro` în exemplul de mai jos), este un nume public pe care serviciul de DNS îl va traduce într-o adresă IP. Dacă se dorește conectarea la un server dar nu se cunoaște numele său DNS, se poate introduce direct IP-ul serverului. Când un client inițiază o conexiune ssh pe un server, trebuie să se conecteze folosind un utilizator existent pe acel server pentru a putea avea acces la interpretorul de comenzi. Parametrul `bogdand` specifică utilizatorul în contul căruia se va intra după stabilirea conexiunii.

```
1 razvan@kiwi:~$ ssh bogdand@securessh.pub.ro
2 Password:
3 Last login: Wed Sep 19 14:37:29 2007 from 86.121.138.243
4 Welcome to the dark side.. we've got cookies!
5
6 bogdand@securessh:~$
```

În rezultatul comenzii de mai sus se poate observa faptul ca s-au mai făcut conexiuni anterioare pe acel server de SSH, fiind oferită atât ora și data ultimei conexiuni, cât și adresa IP de la care s-a realizat conexiunea.

Când un client de ssh inițiază pentru prima dată o conexiune către un server nou, acesta din urmă va trebui să se autentifice către client. Fiecare server de SSH are un identificator unic (*host key*) cu care se autentifică în comunicația cu clienții. Acest identificator este implementat prin chei publice și chei private. La primirea unei conexiuni de la un client, serverul oferă cheia sa publică, făcând astfel posibilă autentificarea. La acceptarea cheii publice de la server, clientul adaugă această cheie în fișierul `known_hosts`. Astfel, următoarea conexiune pe care acest client o va face către server nu va mai necesita transferul cheii publice, aceasta existând deja stocată pe client.

```
1 razvan@kiwi:~$ ssh bogdand@82.77.5.245
2 The authenticity of host '82.77.5.245 (82.77.5.245)' can't be established
  .
3 RSA key fingerprint is 11:94:89:c3:e5:fe:c3:6c:2e:ab:f6:e2:bd:6d:bb:e7.
4 Are you sure you want to continue connecting (yes/no)? yes
5 Warning: Permanently added '82.77.5.245' (RSA) to the list of known hosts
  .
6 Password:
```

```
Have a lot of fun...

bogdand@server:~$
```

Un utilitar pentru sistemele Windows ce permite conectarea în linie de comandă la un server de SSH este aplicația PuTTY. PuTTY este disponibil gratuit și este un client pentru protocolul SSH, telnet, rlogin, dar poate fi folosit și în comunicațiile *TCP raw*. O privire de ansamblu asupra opțiunilor oferite de clienții importanți de SSH poate fi găsită online[1].

Utilizarea SSH pentru rularea de comenzi la distanță

Sintaxa completă a utilitarul **ssh** permite precizarea unei liste de parametri, a utilizatorului și a adresei destinație, precum și a unei comenzi ce va fi rulată după stabilirea sesiunii SSH. Dacă nu este precizată o comandă se va rula un interpretor de comenzi (cel mai adesea `/bin/bash`).

În exemplul de mai jos sunt rulate local două comenzi separate prin ";". Apoi se rulează aceleași comenzi (de data aceasta protejate între ghilimele), rezultatul afișat fiind cel al executării lor pe stația destinație, după autentificarea cu utilizatorul `bogdand`. Se observă din exemplu că autentificarea se realizează fără a interoga utilizatorul, aceasta fiind rezultatul unei autentificări pe bază de chei (vezi secțiunea 10.5.2):

```
rrazvan@apple:~$ hostname; pwd
apple
/home/rrazvan

rrazvan@apple:~$ ssh bogdand@141.85.99.5 "hostname;  pwd"
kiwi
/home/users/bogdand

rrazvan@apple:~$
```

Copierea fișierelor la distanță

Efectuarea transferului de date se poate face prin utilitarul **scp**, ce face parte din pachetul `ssh`. Ideea din spatele **scp** este de a folosi canalul securizat oferit de **ssh** pentru a face transfer de date în format text sau binar între client și server. Sintaxa **scp** este asemănătoare comenzii tradiționale **cp**:

```
scp sursa destinatie
```

Dacă se dorește transferul mai multor fișiere sau directoare, destinația trebuie să fie un director existent. Pentru a efectua o copiere de fișier de pe client, în directorul `/home/bogdand/work/` de pe serverul cu adresa IP 82.77.5.245, comanda este:

```
razvan@kiwi:~$ scp CiscoTFTP.exe bogdand@82.77.5.245:~/work
Password:
CiscoTFTP.exe                                 100% 1296KB 129.6KB/s   00:10

razvan@kiwi:~$
```

[1] http://en.wikipedia.org/wiki/Comparison_of_SSH_clients

Pentru platforma Windows o aplicație *open source* ce poate fi folosită pentru partajarea de fișiere peste o conexiune SSH este WinSCP. Aceasta oferă o interfață grafică pentru transferul de fișiere prin SCP, SFTP sau FTP.

Serverul de SSH

Pentru a putea permite conexiuni SSH spre stația locală, trebuie instalat serverul de OpenSSH (pentru Ubuntu pachetul se numește `openssh-server`). Fișierul de configurare pentru serverul de SSH se află în `/etc/ssh/sshd_config`, și conține directive standard precum: portul pe care se acceptă conexiuni SSH sau opțiunea de a putea rula aplicații X peste conexiunea criptată de SSH.

```
1  razvan@kiwi:/etc/ssh$ cat sshd_config
2  [...]
3   Port 22
4   X11Forwarding yes
5  [...]
```

Autentificarea cu chei

Pentru o securitate sporită a utilizării SSH, se poate folosi autentificarea cu chei publice și chei private. În cazul standard al folosirii SSH, fiecare stație păstrează o asociere între utilizator și parolă în mod automat. Dacă se dorește însă utilizarea criptării bazată pe chei, devine obligația utilizatorului de a crea această asociere manual. Pentru a genera o pereche de chei se poate folosi utilitarul **`ssh-keygen`** instalat odată cu pachetul OpenSSH.

```
1  razvan@kiwi:~$ ssh-keygen -t rsa
2  Generating public/private rsa key pair.
3  Enter file in which to save the key (/home/waters/.ssh/id_rsa):
4  Enter passphrase (empty for no passphrase):
5  Enter same passphrase again:
6  Your identification has been saved in /home/waters/.ssh/id_rsa.
7  Your public key has been saved in /home/waters/.ssh/id_rsa.pub.
8  The key fingerprint is:
9  c1:45:40:92:b6:f6:e5:0b:72:23:73:ff:88:58:27:d0 waters@waters-laptop
10
11 razvan@kiwi:~$
```

Odată generate, cele două chei sunt stocate în mod implicit în două fișiere: `id_rsa` (cheia privată) și `id_rsa.pub` (cheia publică). Cheia publică trebuie să ajungă pe stația unde se află serverul de SSH, iar cheia privată va rămâne local pe stația unde au fost generate cheile.

În momentul creării cheilor, utilizatorului i se va cere un șir de caractere (*passphrase*) care este asociat cu perechea de chei, și care va fi cerut ulterior, în locul parolei. Acest șir poate fi vid, caz în care nu se va mai cere intervenția utilizatorului în cazul utilizării cheilor pentru autentificare. Folosirea unui *passphrase* vid devine deosebit de utilă pentru automatizarea unor acțiuni (prin scripturi).

Pentru a copia cheia publică pe stația aflată la distanță se poate folosi un utilitar dedicat, **`ssh-copy-id`**, specificând ca parametru adresa serverului SSH.

```
1 razvan@kiwi:~$ ssh-copy-id bogdand@82.77.5.245
```

La următoarea încercare de conectare prin SSH, singura diferență va fi cerea *passphrase*-ul (dacă a fost introdus la generarea cheilor) în locul unei parole de cont, însă, din punct de vedere al securității, utilizarea serviciului va fi mai sigură.

9.2.3 Utilitarul wget

wget este un utilitar gratuit (din suita GNU Project), creat pentru a facilita o descarcare simplă și eficientă a resurselor de pe Web. Aceste resurse pot fi obținute cu **wget** prin două dintre cele mai folosite protocoale din Internet: HTTP și FTP. Numele utilitarului este obținut de fapt printr-o alăturare semantică a World Wide Web (W) și a metodei `HTTP GET`.

Principalul avantaj al **wget** este faptul că execută o **descărcare neinteractivă** a fișierelor. Într-o aplicație din mediul grafic (Firefox, Filezilla), o descărcarea uzuală necesită prezența pe sistem a utilizatorului. Cu alte cuvinte, acesta trebuie să fie logat pentru a se putea efectua operația de descărcare, sau pentru a avea posibilitatea de a reactiva o descărcarea întreruptă de către server.

În contrast, folosind **wget**, utilizatorul poate să se autentifice pe sistem, să pornească descărcarea, și să se deautentifice, **wget** continuând să ruleze până la terminarea sarcinilor neîncheiate. Mai mult, dacă conexiunea către server este întreruptă, utilitarul va stoca fișierele descărcate până în acel moment pe hard-disc, va salva parametrii sesiunii și adresa serverului și va putea continua sarcina de descărcare în momentul în care legătura cu serverul se reface și resursa este din nou accesibilă.

Prezentând un astfel de avantaj evident față de clienții de descărcare din mediul grafic, **wget** a fost rapid adoptat de marea majoritate a utilizatorilor care aveau o conexiune înceată la Internet și care nu puteau petrece autentificați pe sistem timpul necesar unei descărcări interactive. Pe măsură ce Web-ul a evoluat **wget** adăugat noi funcționalități.

Un avantaj notabil al utilitarului **wget** este capabilitatea de descărcare și de păstrare pe client a structurii de directoare de pe server, fiind de asemenea posibilă descărcarea unui site cu indicarea nivelului de adâncime până la care să fie descărcată structura de directoare.

Acest tip de descărcare, numit **descărcare recursivă**, a fost extins până la posibilitatea de a descărca pe client site-uri întregi cu toate paginile și legăturile dintre acestea. Considerând situația unei rețele locale, descărcarea unui site pe mașina server din rețea (proces numit și oglindirea site-ului), ar face implicit posibilă accesarea acelui site în mod *offline* de către toți clienții din rețeaua locală.

Modul de apelare al utilitarului în linia de comandă

Pentru a descărca un fișier prin HTTP se poate folosi comanda:

```
1 razvan@kiwi:~$ wget -c http://ccnp.catc.ro/resources/CiscoTFTP.exe \&
2
3 --15:57:06--  http://ccnp.catc.ro/resources/CiscoTFTP.exe
```

```
4             => 'CiscoTFTP.exe'
5
6  Resolving ccnp.catc.ro... 141.85.37.2
7  Connecting to ccnp.catc.ro|141.85.37.2|:80... connected.
8  HTTP request sent, awaiting response... 200 OK
9  Length: 1,327,497 (1.3M) [application/x-msdos-program]
10
11 100%[====================================>] 1,327,497      2.66M/s
12
13 15:57:07 (2.66 MB/s) - 'CiscoTFTP.exe' saved [1327497/1327497]
```

Opțiunea `-c` asigură continuarea descărcării în cazul întreruperii conexiunii cu serverul, iar simbolul `&` specifică rularea utilitarului în *background*.

Descărcarea unui site pe discul local pentru a face posibilă accesarea sa *offline* este exemplificată prin comanda următoare:

```
1  razvan@kiwi:~$ wget --convert-links -r http://www.gnu.org/ -o gnulog
2
3  razvan@kiwi:~$ ls
4  gnulog  www.fsf.org  www.gnu.org
```

Parametrul `--convert-links` transformă legăturile (linkurile) din site pentru ca site-ul să poată fi vizionat *offline*.

Descărcarea tuturor legăturilor de pe siteul `www.yahoo.com` cu o adâncime de 1 se face prin:

```
1  razvan@kiwi:~$ wget -r -l1 www.yahoo.com
```

9.3 Email – Poșta electronică

> **Poșta electronică** (electronic mail), abreviat e-mail sau email, este o metodă de compunere, transmitere și recepție a mesajelor peste sisteme de comunicație electronică.

Ideea de mesagerie electronică datează din anul 1971, când Ray Tomlinson dezvolta prima aplicație de email pentru ARPANET. Aceasta era formată din două programe: SNDMSG, utilizat pentru a transmite mesaje, respectiv READMAIL, folosit pentru citirea mesajelor. În anul 1972, comenzile `MAIL` și `MLFL` au fost adăugate programului FTP. Aceasta a fost modalitatea de transmitere a mesajelor în rețeaua ARPANET până la începutul anilor '80, când a fost dezvoltat protocolul SMTP. În scurt timp, serviciul de email a crescut popularitatea ARPANET-ului și a devenit aplicația principală a acestuia.

Serviciul de email a oferit la început doar transferul de mesaje text între utilizatori aflați în rețele diferite. Ulterior serviciul a fost extins și la transmiterea de fișiere binare. Standardul de codificare a informației binare într-un mesaj e-mail se numește MIME (*Multipurpose Internet Mail Extensions*).

9.3.1 Arhitectură și funcționare

Pentru livrarea mesajelor se folosește un mecanism de adresare ce apelează atât ierarhia de nume din Internet (serviciul de DNS) pentru a preciza serverul, cât și informații relevante local, precum numele utilizatorului.

O adresă de email conține două tipuri de informații separate prin simbolul @ (citit la sau *at* sau *a-round*):

- numele utilizatorului care dorește transmiterea sau recepția mesajului;
- numele (DNS) al server-ului care transmite/recepționează mesajul.

Pe baza adresei de email destinație serviciul de poștă electronică livrează mesajele de la clienții de email, folosind servere de email, către căsuța poștală asociată cu adresa de email destinație.

> **Căsuță poștală** este o intrare în sistemul local de fișiere utilizată pentru stocarea mesajelor de poștă electronică corespunzătoare unui utilizator.

Componenta esențială a serviciului de poștă electronică o reprezintă protocolul **SMTP** (*Simple Mail Transfer Protocol*). Acest protocol oferă mecanisme pentru preluarea mesajelor de la clienți de email și livrarea lor pe serverele de email destinație (ca atare pentru trimiterea mesajelor).

Odată cu dezvoltarea sistemelor de operare pentru calculatoarele personale, a apărut și nevoia preluării noilor mesaje din căsuța poștală pe stația utilizatorului. Acest lucru este realizat printr-un nou serviciu ce asigură ridicarea email-ului și sincronizarea căsuței poștale de pe server cu cea de pe stația de lucru.

Pentru ridicarea mesajelor de pe serverul de email cele mai folosite protocoale sunt **IMAP** (*Internet Message Access Protocol*) și **POP3** (*Post Office Protocol version 3*). Pentru oferirea acestui serviciu va trebui pornit un daemon de IMAP sau POP3, în plus față de serverul de SMTP.

Marea majoritate a clienților de email integrează pe lângă componenta de client SMTP (folosită pentru transmiterea mesajelor) și componente de client IMAP și POP3 pentru interogarea căsuței poștale de pe serverul de email.

O deosebire importantă a IMAP față de POP3 o reprezintă extinderea posibilităților de prelucrare a mesajelor din căsuța poștală oferite de protocolul IMAP. Folosind IMAP mesajele nu sunt descărcate de pe server decât la vizualizarea lor, existând totodată și opțiunea descărcării doar a unei părți din mesaj.

Spre deosebire de POP3, unde toate mesajele se găsesc pe server într-o singură căsuță poștală (*inbox*), iar directoarele nu se pot crea decât în aplicația client, protocolul IMAP permite crearea de directoaret pe server direc. Utilizând comenzi IMAP, aplicația client poate folosi filtre pentru mutarea mesajelor dintr-un director în altul, fără să fie nevoie măcar ca *mailbox*-urile să fie situate pe același server.

Rezumând, arhitectura serviciului de email include două componente principale: un serviciu pentru primirea și trimiterea mesajelor și un serviciu pentru ridicarea mesajelor de pe serverul de email.

Trimiterea unui mesaj

Pentru a exemplifica modul în care componentele prezente mai sus interacționează, vom considera următoarea situație: Dan are adresa de mail dan@a.org și dorește să-i transmită un mesaj lui Bogdan care are adresa de mail bogdan@b.org.

Pentru livrarea unui mesaj în căsuța poștală asociată contului bogdan@b.org, pașii sunt prezentați în figura 9.6.

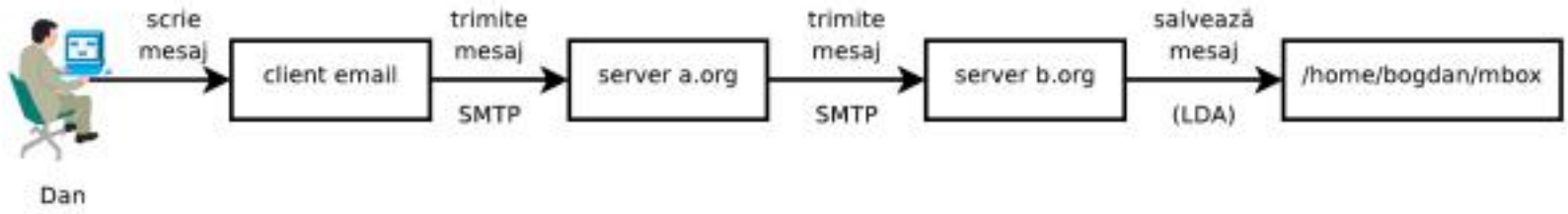


Figura 9.6: Livrarea unui mesaj

Dan va deschide un client de email în care va edita mesajul. Când Dan va da comanda de trimitere a mesajului, clientul de email va iniția o conexiune SMTP către serverul de mail local. Pentru stabilirea conexiunii, serverul va cere autentificarea utilizatorului Dan. În general, clientul de mail va salva numele de utilizator și parola și va realiza autentificarea conexiunii SMTP în mod transparent pentru utilizator. După autentificare, clientul de email va formata mesajul conform protocolului SMTP și îl va trimite către server.

Serverul de mail local va analiza apoi adresa de email destinație, pe baza căreia va determina domeniul de nume în care trebuie livrat mesajul (în cazul exemplului: b.org). Va iniția o cere de rezolvare de nume (cerere DNS) pentru a determina adresa serverului responsabil cu preluarea mailului pentru domeniul b.org (acest server se numește *mail exchanger*). După determinarea serverului de mail pentru b.org, serverul local va trimite mesajul folosind protocolul SMTP.

Serverul b.org va stoca mesajul în căsuța poștală a utilizatorului `bogdan`. La nivelul acestui pas se poate interpune o aplicație de distribuție a mesajelor denumită generic LDA (*Local Delivery Agent*). O astfel de aplicație poate realiza filtrarea mesajelor, livrarea anumitor mesaje în alte directoare, oprirea spamurilor etc. În Linux cea mai folosită aplicație de distribuție a mesajelor este oferită de pachetul `procmail`.

Ridicarea mesajelor

Pentru a ridica mesajul din căsuța poștală vor fi urmați pașii din figura 9.7:

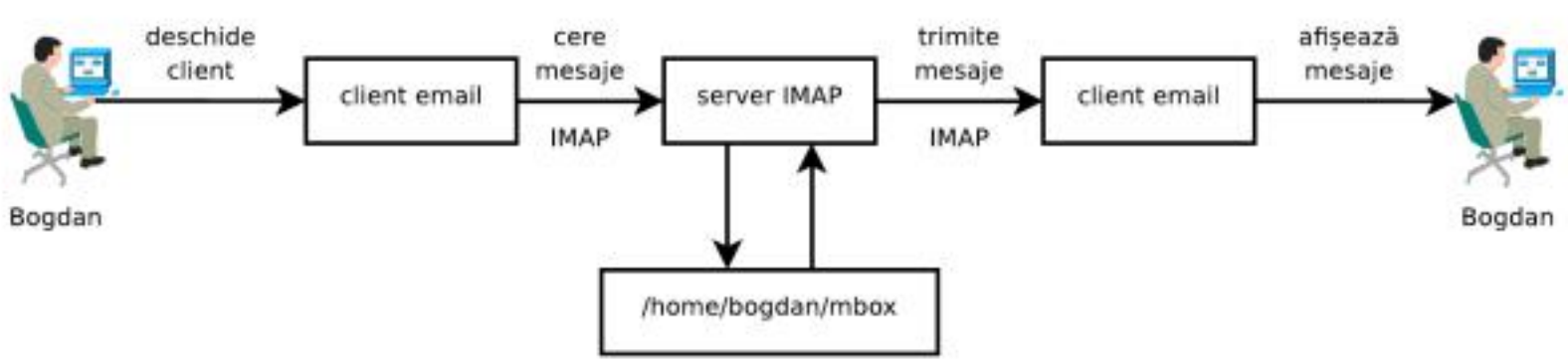


Figura 9.7: Ridicarea mesajelor de pe server

Bogdan va deschide clientul de mail, iar acesta va iniția o sesiune de IMAP către serverul de mail. După autentificare, serverul va deschide căsuța poștală a utilizatorului Bogdan. Mesajele ridicate din căsuța poștală în sesiunile anterioare de IMAP au fost marcate, astfel serverul va selecta doar noile mesaje (ce nu sunt marcate) și le va transmite prin sesiunea de IMAP către clientul de email. Înainte de a încheia sesiunea, clientul de email va informa serverul asupra listei mesajelor ce au fost șterse din căsuța poștală aflată pe stația utilizatorului, permițând astfel serverului să sincronizeze conținutul căsuței poștale locale cu cea aflată pe stație.

După ridicarea mesajelor de pe server, căsuța poștală poate fi consultată inclusiv *offline*, folosind clientul de email.

Folosirea webmail pentru citirea și transmiterea mesajelor

O altă metodă disponibilă utilizatorilor pentru citirea și trimiterea mesajelor este folosirea unui server de web pentru interfața cu serverul de SMTP, serviciu rezultat fiind denumit **webmail**.

Folosind exemplul anterior, mesajul livrat prin SMTP a fost stocat în căsuța poștală a utilizatorului Bogdan. Utilizatorul va lansa un client web (un broswer) și va iniția o conexiune HTTP către serverul de web ce rulează pe același sistem cu serverul de SMTP, responsabil cu primirea mesajelor pentru domeniul b.org. Serverul de web va oferi o interfață ce permite operații direct pe căsuța poștală aflată pe server: ștergerea/mutarea mesajelor, marcarea lor, crearea de directoare etc.

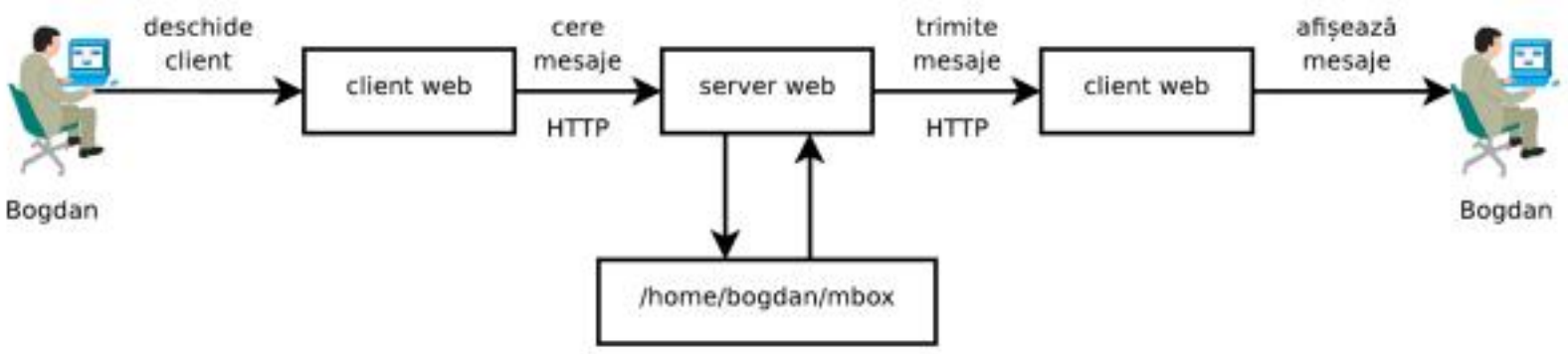


Figura 9.8: Citirea mesajelor folosind webmail

9.3.2 Clienți de email

Așa cum s-a prezentat mai sus, clienții de email sunt utilitare care permit descărcarea de mesaje de pe un server de email și stocarea lor local. În mod real, există două situații în care este necesară folosirea lor: atunci când contul de email nu poate fi vizualizat printr-o interfață de tip web, sau atunci când conexiunea la Internet este slabă sau instabilă, și este de preferat salvarea corespondenței pe computerul personal pentru a fi ușor accesată și ulterior. Există mai mulți clienți de email, cei mai importanți fiind Microsoft Outlook (varianta inclusă în distribuția de Windows se numește Outlook Express) și Mozilla Thunderbird.

Un avantaj al clienților de email este stocarea centralizată a mesajelor primite pe conturi de email diferite. În cele ce urmează, ne vom concentra asupra Mozilla Thunderbird,

deoarece este gratuit și disponibil pentru toate sistemele majore de operare (Windows, Linux, Mac OS).

Configurarea unui client de email presupune în primul rând crearea unui cont pentru fiecare adresă de email ce se dorește administrata cu Thunderbird. Sunt folosite două protocoale diferite: SMTP pentru a trimite mesaje și POP3 pentru a primi mesaje. Setarea unui cont presupune introducerea adresei de email, parolei asociate, adresa serverului de SMTP și a serverului de POP3. Se poate opta pentru salvarea parolei într-un fișier de configurare, sau pentru introducerea ei manuală de fiecare data când se va stabili o conexiune la unul dintre serverele de mail. În cazul în care doar unul dintre cele două servere este setat corespunzător, se vor putea doar descărca (în cazul POP3) sau trimite mesaje (respectiv SMTP).

Unul dintre avantajele Thunderbird este integrarea cu serviciul de email oferit de Google, numit Gmail. Pentru setarea unui cont de Gmail, trebuie parcurși următorii pași: se activează pe site-ul Gmail opțiunea de a folosi un server de POP3, apoi din Thunderbird se creează un nou cont, se alege *Gmail account* și se introduce adresa de email. Restul setărilor se efectuează automat.

Pentru a trimite/primi corespondență, se folosește butonul *Get Mail*, comandă ce sincronizează baza de date locală cu cea de pe server. În consecință, un email care este scris poate fi trimis instantaneu sau poate fi stocat local și trimis mai târziu, când va exista conectivitate la Internet. Thunderbird se sincronizează în mod automat cu serverul la intervale regulate de timp.

Thunderbird oferă o gamă de servicii standard: memorarea adreselor de email, sortarea emailurilor după, data primirii, subiect, numele expeditorului, importanță (crescător sau descrescător). Emailurile pot fi grupate pe threaduri (toate mesajele care fac parte din aceeași discuție), facilitate deosebit de folositoare în cazul mesajelor primite de la o listă de discuții. De asemenea pentru localizarea unui mesaj clientul oferă un motor de căutare ce poate folosii criterii ce vor analiza: adresa sursă, adresa destinație, subiectul mesajului sau conținutul său.

Pentru o mai buna organizare a corespondenței, se poate crea o structură de directoare și gruparea mesajelor în mod logic, în opoziție cu stocarea tradițională a tuturor email-urilor în căsuța poștală (*inbox*). Aplicarea de filtre permite selectarea emailurilor pe bază de criterii avansate (ce includ adresă, nume expeditor, subiect, conținut) și efectuarea de acțiuni asupra lor (mutarea în diferite directoare, ștergerea lor, marcarea lor), facilitate extrem de utilă în cazul unui număr mare de mesaje zilnice.

O limitare în folosirea unui soluții mixte bazate pe Thunderbird și Gmail o reprezintă nesincronizarea marcării mesajelor citite între cele două aplicații.

9.3.3 Securitatea serviciului de email

Cea mai întâlnită problemă asociată cu poșta electronică o reprezintă primirea de spam (mesaje nesolicitate). Din păcate, traficul de **spam** a depășit 90% din traficul SMTP în Internet.

Pentru a reduce traficul de spam în primul rând trebui ca serverul de email (SMTP) să nu accepte să trimită mesaje de la orice client SMTP din Internet. Majoritatea destinațiilor

de mail vor verifica pentru fiecare mesaj dacă nu a fost trimis folosind un server SMTP de tip *open mail relay* (server ce acceptă să trimită mesaje pentru orice client), caz în care vor considera respectivul mesaj drept spam.

Traficul de spam poate fi controlat și pe serverul destinație înainte de a fi preluat de aplicația client. Reducerea traficului de spam pe serverul de SMTP poate fi făcută prin implementarea a trei tipuri de liste:

- **Liste albe** – mesajele trimise de un server conținut într-o astfel de listă sunt considerate automat mesaje legitime
- **Liste gri** – mesajele primite de la servere necunoscute sunt, pentru o perioadă de timp, respinse. Dacă serverul sursă încercă retransmiterea mesajului și după expirarea acestei perioade, mesajul va fi considerat legitim. Această metodă se bazează pe faptul că serverele de spam nu vor încerca retransmiterea mesajelor.
- **Liste negre** – mesajele primite de la un astfel de server sunt etichetate ca spam

O altă clasă de aplicații de filtrare a traficului de spam sunt integrate în LDA (*Local Delivery Agent*). Aceste aplicații în general folosesc un set extins de criterii pentru a calcula probabilitatea ca un mail să fie spam.

Filtrarea traficului de spam poate fi realizată inclusiv de clientul de mail. Cu toate acestea, chiar și în cazul folosirii simultane a tuturor metodelor de control al traficului nedorit, mesajele de spam nu pot fi total eliminate.

O altă problemă importantă a serviciului de email este **absența unui mecanism de autentificare a expeditorului**. Există extensii ale protocolului SMTP ce rezolvă această problema, dar care nu se bucură de o popularitate însemnată. Singura soluție cu adevărat eficientă în prevenirea atacurilor bazate pe asumarea unui false identități (numite atacuri phising) este instruirea utilizatorilor asupra limitărilor protocolului SMTP.

9.4 WWW

World Wide Web sau WWW este, probabil, cel mai cunoscut și utilizat serviciu din Internet. WWW-ul se prezintă ca un set de resurse interconectate prin intermediul de legături (*hyperlink*-uri). Aceste resurse sunt denumite pagini web și conțin *hypertext*, imagini, filme. *Hypertext*-ul este în esență text simplu, la care s-a adăugat o formatare standardizată, ce specifică utilizarea de *hyperlink*-uri. Accesul la resurse se realizează prin intermediul unui navigator (*browser*). O pagină web va conține, de obicei, legături (*hyperlink*-uri) către alte pagini (resurse), justificând denumirea de Web.

World Wide Web-ul este invenția lui **Sir Tim Berners-Lee** din 1989, una din personalitățile marcante în dezvoltarea Internet-ului. Sir Tim Berners-Lee este directorul World Wide Web Consortium (W3C), organizația care coordonează dezvoltarea WWW.

În ultimii ani, WWW-ul a cunoscut o dezvoltare mult mai amplă ajungându-se la folosirea expresiei **Web 2.0** pentru a simboliza serviciile web de nouă generație oferite de diversele comunități web. În același timp, WWW-ul evoluează către ceea ce Sir Tim Berners Lee a denumit **Semantic Web**, adică o extensie a WWW-ului în care conținutul poate fi exprimat în limbaj natural și utilizat de agenți specifici de analiză.

9.4.1 Tehnologiile de bază ale WWW

WWW-ul își datorează existență celor trei tehnologii utilizate de Sir Tim Berners-Lee. Acestea sunt URL, HTTP și HTML și sunt prezentate în continuare.

URL

URL înseamnă *Uniform Resource Locator* și este un nume într-o sintaxă bine precizată pentru a identifica resursele din Internet. URL este de fapt un sinonim pentru URI (*Uniform Resource Identifier*).

În mod tipic, un URL/URI este introdus în bara de adrese a unui navigator și este folosit pentru accesarea unei resurse (a unei pagini web) din Internet. Un navigator va "decodifica" URL-ul și va accesa resursa dorită. Astfel, în cazul în care se accesează URL-ul `http://www.example.com` se face o cerere HTTP către server-ul `example.com`. Pe de altă parte, URI-ul `mailto:bogdan@example.com` va însemna deschiderea unui client de e-mail pentru compunerea unui mesaj către bogdan@example.com.

Un URL are o sintaxă bine specificată folosită pentru identificarea stației din Internet, a serviciului folosit de aceasta și a resursei pe care o poate pune la dispoziție acest serviciu. Astfel, în cazul URL-ului `http://www.example.com/page.html` stația care deține resursa este server-ul `www.example.com`. O cerere DNS va fi folosită pentru translatarea numelui într-o adresă IP. Șirul `http://` este folosit pentru identificarea serviciului; în cazul de față acesta este serviciul de Web care folosește protocolul HTTP și portul implicit `80`. Resursa cerută de pe server este `/page.html`. Drept urmare, în urma introducerii URL-ui în bara de adrese a navigatorului, acesta va face o cerere HTTP către stația identificată de numele `www.example.com` pentru pagina `/page.html` de pe server. Aceasta va fi afișată apoi clientului (în interiorul *browser*-ului).

Exemplul de URL prezentat mai sus este unul simplificat. Astfel, sintaxa completă pentru un URL este:

```
1 protocol://server:port/cale/catre/resursa
```

- câmpul `protocol` specifică, evident, protocolul folosit pentru comunicație. Acesta poate fi `ftp`, `http`, `scp` etc.; dacă acest câmp lipsește, un browser va folosi implicit `http`;
- câmpul `server` reprezintă numele sau adresa IP a stației cu care se dorește realizarea unei comunicații;
- câmpul `port` este portul utilizat (acesta poate lipsi), caz în care se folosește portul implicit al serviciului;
- `/cale/catre/resursa` reprezintă calea către resursa aflată pe server; acesta va folosit această cale pentru a oferi respectiva resursă clientului; de obicei, resursa este o intrare în sistemul de fișiere al server-ului.

Sintaxa URL-ului poate fi complicată. Astfel, un URL tip ftp poate asigura inclusiv autentificarea pe server. Folosirea URL-ului `ftp://bogdan:p4r0l48un4@example.com/sources.tgz` înseamnă

autentificarea utilizatorului `bogdan` cu parola `p4r0l48un4` pe serverul `example.com`, folosind `ftp` pentru descărcarea fișierului `sources.tgz`.

HTTP

HTTP (*Hypertext Transfer Protocol*) este protocolul fundamental al World Wide Web-ului. Deși navigatoarele pot folosi alte protocoale în afara HTTP, acesta rămâne cel mai important mecanism de comunicare.

Similar celorlalte servicii din Internet, serviciul Web (HTTP) funcționează ca sistem client-server. Clientul Web este navigatorul (browser-ul) folosit pentru cererea unei resurse prin intermediul unui URL. Portul implicit utilizat de HTTP este 80.

Protocolul HTTP este un protocol de tip întrebare răspuns (*request – response*). Clientul Web (navigatorul) face o cerere către un server, după care acesta îi trimite un răspuns și resursa cerută. În exemplul de mai jos, telnet este utilizat pentru a efectua o cerere HTTP către `google.ro`:

```
1 razvan@anaconda:~$ telnet google.ro 80
2 Trying 72.14.221.104...
3 Connected to google.ro.
4 Escape character is '^]'.
5 GET / HTTP/1.0
6 HTTP/1.0 302 Found
7 Location: http://www.google.ro/
8 Cache-Control: private
9 [...]
```

După realizarea conexiunii pe portul 80, clientul trimite cererea `GET / HTTP/1.0`. Aceasta înseamnă citirea directorului rădăcină al serverului folosind HTTP versiunea 1.0. Răspunsul server-ului este `HTTP/1.0 302` Found urmat de pagina efectivă. Dacă pagina dorită nu se află pe server, răspunsul server-ului va fi `Error 402 Page not Found`. Acest răspuns nu înseamnă însă că serverul nu funcționează, caz în care mesajul afișat de client ar fi `Server not Found`.

O altă caracteristică importantă a protocolului HTTP este faptul că este un protocol neorientat pe conexiune (*stateless*). Acest lucru înseamnă că după ce clientul trimite o cerere și îi sosește răspunsul conexiunea este încheiată. Pentru o nouă resursă trebuie realizată o altă conexiune. Opus este protocolul FTP care este un protocol cu conexiune persistentă. Petru că este nevoie ca anumite informații să fie persistente de-a lungul mai multor conexiuni s-au găsit diverse soluții: cookie-uri pe client, sesiuni pe server, variabile HTTP etc.

Un alt neajuns al HTTP este lipsa unui suport puternic de securitate. Acest lucru este corectat prin folosirea protocolului **HTTPS** care folosește comunicație criptată. Majoritatea sistemelor de webmail de astăzi folosesc HTTPS pentru a asigura confidențialitatea datelor.

HTML

HTML (*Hypertext Markup Language*) este un limbaj folosit pentru a descrie conținutul unei pagini Web. HTML folosește **marcaje** (tag-uri) pentru a defini modul în care un

element dintr-o pagină Web este afișat de un browser.

O pagină simplă Web descrisă în format HTML este următoarea:

```
razvan@anaconda:~/public_html$ cat main.html
<html>
        <head>
                <title>Pagina mea</title>
        </head>

        <body>
                <h1>Antet</h1>

                Text simplu
                <strong>Text aldin</strong>
                <em>Text cursiv</em>
        </body>
</html>
```

Un marcaj încadrează de obicei un text. La început, marcajul este `<nume_marcaj>` și se încheie cu `</nume_marcaj>`. Marcajele prezente în această pagină sunt:

- **html**, pentru definirea documentului HTML
- **head** prezintă informații despre pagină, spre exemplu titlul (marcajul title)
- **h1** definește un antet de nivel 1
- **strong** definește un text aldin (bold)
- **em** definește un text cursiv (italic)

Un client Web va interpreta o pagină HTML și o va afișa utilizatorui conform marcajelor existente.

XHTML

XHTML (Extensible Hypertext Markup Language) este un standard și un limbaj de marcare care are aceleași posibilitați de exprimare ca și HTML însă se conformează sintaxei mult mai stricte a XML. Urmând o sintaxă mult mai strictă, un document XHTML este mult mai ușor de interpretat, spre deosebire de un document HTML care necesită un parser complex.

Motivația utilizării XHTML o reprezintă creșterea numărului de dispozitive care nu au capacitatea de procesare a unui document HTML complex. Sintaxa strictă a XHTML permite o prelucrare mult mai rapidă a unei pagini Web de către client.

Adoptarea XHTML se realizează într-un ritm inegal datorită lipsei de implementare în navigatoare a caracteristicilor noi pe care standardul le aduce.

Versiunea curentă de XHTML este 1.1. Versiunea 2.0 a standardului este încă în dezvoltare.

9.4.2 Funcționarea serviciului

După cum s-a precizat, serviciul de Web funcționează în sistem client-server și folosește protocolul HTTP pentru comunicare. Funcționarea serviciului este sistematizată în figura 9.9, în care sunt prezentați un set de pași urmați tipic în obținerea de către client, a unei resurse aflate pe un server Web (de obicei o pagină Web).

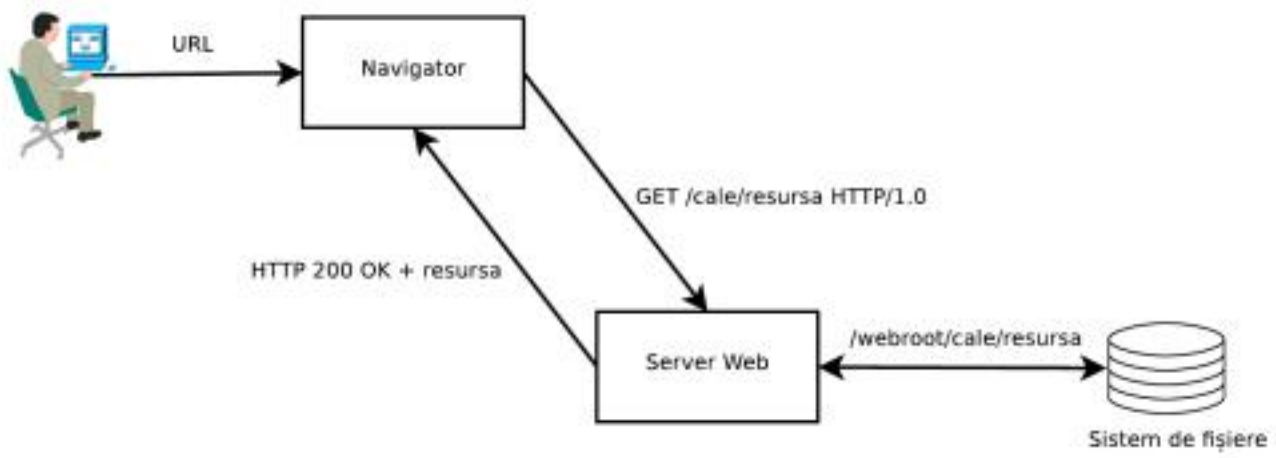


Figura 9.9: Funcționarea serviciului de Web

În primă fază utilizatorul va folosi un client Web (navigator) și va introduce în bara de adrese a acestuia un URL care identifică resursa dorită.

Navigatorul identifică din URL sistemul unde trebuie să se conecteze și resursa cerută. De obicei se va genera și o cerere DNS pentru a afla adresa IP asociată serverului. Navigatorul se conectează la server și, în cazul protocolului HTTP, formulează o cerere de forma `GET /cale/resursa HTTP/1.0`.

Cererea ajunge la serverul Web. Acesta trebuie să ofere clientului resursa cerută. Resursa este, de obicei, o intrare în sistemul de fișiere (pagină web, imagine, film etc.). Drept urmare serverul Web va trebui să localizeze resursa. Orice server web are configurat un director rădăcină specializat, denumit și **webroot**. Orice resursă este obținută din acest director. Drept urmare, calea completă în sistemul de fișiere este obținută prin concatenarea căii din cererea HTTP la *webroot*, rezultând intrarea `/webroot/cale/ resursa`.

După ce obține resursa obținută, serverul va trebui să o transmită clientului. Pentru aceasta el emite un răspuns `HTTP 200 OK` și atașează resursa.

Clientul primește resursa de la server și, dacă este vorba de o pagină Web, interpretează codul (X)HTML și afișează rezultatul utilizatorului. Conexiunea se închide.

9.4.3 Servere Web

Un **server Web** este responsabil cu oferirea de resurse din sistemul local de fișiere către clienți (navigatoare) prin intermediul protocolului HTTP. Exemple de servere Web sunt Apache HTTP Server, Microsoft IIS, Sun Web Server, Zeus Web Server. Apache HTTP Server este cel mai folosit server Web din Internet.

În general, un server Web prezintă un fișier de configurare prin intermediul căruia îi poate fi alterat comportamentul. În cazul Apache HTTP Server versiunea 2, acest fișier

de configurare este `/etc/apache2/apache2.conf`. Alterarea acestui fișier de configurare permite schimbarea portului pe care acesta ascultă conexiuni (implicit 80), a webroot-ului (implicit `/var/www`) și altele.

Un server Web va primi o cerere HTTP în forma `GET /cale/catre/resursa HTTP/ 1.0` și va identifica din sistemul local de fișiere intrarea asociată resursei. Fiecare server are un director rădăcină (denumit *webroot*) unde este căutată resursa pentru a fi oferită clientului. În cazul serverului Apache, în urma unei cereri `GET /cale/catre/resursa HTTP/1.0` va trimite clientului fișierul identificat de `/var/www/cale/catre/resursa` (sau `/usr/local/Apache2/htdocs`).

9.4.4 Clienții Web

Clienții Web se numesc **navigatoare** sau **browser**-e. Un client Web se va conecta la un server Web pentru a solicita o resursă. Resursa este specificată de utilizator într-un URL. Clientul Web transmite o cerere HTTP către server solicitând resursa clientului. De obicei această resursă este o pagină Web. Codul HTML este interpretat apoi de navigator și afișat utilizatorului.

Cele mai utilizate tipuri de navigatoare sunt Internet Explorer, Mozilla Firefox, Opera și Safari. Internet Explorer este cel mai utilizat navigator cu o pondere de 80-85% urmat de Mozilla Firefox cu o pondere de 10-15%. Celelalte navigatoare au ponderi mult mai mici. Există și navigatoare în linie de comandă, cum ar fi **`lynx`**, **`w3m`**, **`links`**.

Navigatoarele Web reprezintă una dintre cele mai utilizate aplicații din cadrul unui sistem de operare. Competiția dintre acestea a generat ceea ce s-a numit "*browser wars*". Astfel, anii 1994-1998 au fost marcați de competiția dintre Netscape Navigator și Internet Explorer cu victoria celui din urmă. În anul 2002, Internet Explorer era folosit în proporție de circa 92%. Ultimii ani sunt dominați de competiția între Mozilla Firefox și Internet Explorer.

Competiția dintre navigatoare a condus și la apariția unui număr important de caracteristici printre care: navigare tabulară, blocarea mesajelor de tip pop-up, corecție gramaticală, gesturi de mouse, suport pentru diverse protocoale (FTP, IRC, BitTorrent) și pentru standarde (CSS, XHTML, XSLT, RSS).

9.5 Studii de caz

9.5.1 Utilitarul cURL

Utilitarul cURL este aplicație pentru transferul de fișiere ce își propune să ofere suport atât pentru cerințe avansate de securitate, cât și pentru un număr foarte mare de protocoale. Lista protocoalelor implementate de cURL include FTP, HTTP, telnet, LDAP, precum și protocoale orientate spre securitate precum SCP, SFTP, FTPS, HTTPS. Pentru asigurarea securității se pot folosi în cURL numeroase metode de autentificare pe bază de parolă, cât și pe certificate.

Utilitarul cURL ce se execută din linia de comandă, ceea ce permite includerea sa în scripturi. Cel mai adesea cURL este folosit pentru scripting HTTP, în scopul simulării activității utilizatorilor ce folosesc diferite broswere.

Utilitarul cURL implementează numeroase mecanisme specifice clienților web, cele mai multe suplimentare față de **wget**. Spre exemplu oferă gestionarea *cookies* (trimiterea și salvarea locală), navigarea prin pagini de formular, folosirea unui proxy web, urmărirea redirectărilor automate, sau schimbarea semnăturii navigatorului.

Spre exemplu, presupunem că în cadrul unui site se află un formular disponibil la adresa: `test.cs.pub.ro/login.cgi`. În acest formular trebuie completat un câmp de nume și apoi apăsat butonul OK pentru a accesa site-ul. Formularul este scris folosind sintaxa HTML:

```
1 <form method="GET" action="login.cgi">
2     <input type=text name="nume">
3     <input type=submit name=press value="OK">
4 </form>
```

Listing 9.1: Formular de login simplu

Cererea poate fi trimisă folosind cURL astfel:

```
1 razvan@kiwi:~$ curl "test.cs.pub.ro/login.cgi?nume=Razvan\&press=OK"
```

În cazul unui formular ce permite clientului să încarce un fișier se poate folosi cURL cu opțiunea `-F`. Presupunem că formularul este scris astfel:

```
1 <form method="POST" enctype='multipart/form-data' action="upload.cgi">
2     <input type=file name=upload>
3     <input type=submit name=press value="OK">
4 </form>
```

Listing 9.2: Formular de upload simplu

Pentru formularul aflat la adresa `test.cs.pub.ro/upload.cgi` va fi încărcat fișierul `test01` și apoi simulată apăsarea butonului OK:

```
1 razvan@kiwi:~$ curl -F upload=@test01 -F press=OK test.cs.pub.ro/upload.
  cgi
```

Utilitarul cURL oferă suport și pentru schimbarea identității navigatorului. Pentru a simula o cerere către serverul test.com inițiată de un navigator Internet Explorer 7.0 ce rulează pe un sistem Windows 2000 vom folosi comanda:

```
1 razvan@kiwi:~$ curl -A "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT
  5.0)" test.co
```

9.5.2 FTP

FTP (*File Transfer Protocol*) este un protocol foarte răspândit ce oferă posibilitatea transferului de fișiere atât într-o rețea locală cât și prin Internet. Se bazează pe modelul client-server, deci presupune existența a două entități între care se realizează un transfer de fișiere, clientul fiind cel care se conectează (și se autentifică) la serverul de la care poate descărca sau căruia îi poate trimite fișiere.

Ca și în cazul modelului generalizat de client-server, serviciul FTP oferit de către acesta din urmă se reduce la existența unei aplicații, numită "server FTP" ce rulează pe o stație. Similar, clientul rulează o aplicație numită "client FTP" ce se ocupă de managementul conexiunii către server și de traficul de date dintre aceștia. Atât aplicațiile server cât și cele client FTP sunt disponibile, practic, pentru toate platformele existente, dovadă a versatilității și răspândirii protocolului FTP.

Un dezavantaj al protocolului FTP este faptul că acesta transferă atât informațiile de conectare cât și conținutul propriu-zis al fișierelor în mod necriptat, interceptarea unui astfel de trafic putând duce la o interpretare facilă a sa.

Funcționare

Un client FTP ce se conectează la un server trimite acestuia un set de date de autentificare (nume și parolă) care au asociate pe server un o structură de directoare ce sunt puse la dispoziția acelui utilizator. Dacă autentificarea reușește, clientul are dreptul să trimită în continuare comenzi pentru a interacționa cu fișierele la care are acces. Drepturile de acces la fișiere se reduc la cele de listare/creare de directoare, citire, modificare, ștergere de fișiere sau directoare, ceea ce implică un transfer bidirecțional între client și server (clientul poate atât să descarce, cât și să trimită fișiere pe server). În practică, aceste drepturi sunt enunțate în aceeași structură de tip "rwx" ca și pe sistemele UNIX.

Serverul FTP "ascultă" conexiunile pe portul 21 și realizează transferul de comenzi tot pe acesta. Transferul efectiv de fișiere se realizează în paralel, pe portul 20, doar în urma unei conexiuni prestabilite. Porturile rezervate pentru protocolul FTP sunt 21 pentru stabilirea conexiunii și 20 pentru transferul efectiv de date. O variantă securizată și similară o reprezintă SFTP (SSH File Transfer Protocol). O altă limitare a sa este imposibilitatea de a transmite atributele ce țin de data și ora creării/modificării fișierelor, acestea fiind rescrise la destinație. De asemenea, această limitare este eliminată în cazul lui SFTP.

Transferul datelor prin FTP se poate face prin două moduri, selectate manual înaintea inițierii transferului sau determinate automat de către clientul FTP:

- Mod ASCII: folosit pentru transferul fișierelor text; are avantajul de a transmite datele folosind codurile ASCII ale caracterelor, ceea ce permite compresia dinamică a informației, la un raport foarte bun, deci un flux mai puternic de date, dar are dezavantajul de a corupe conținutul oricărui fișier non-text.
- Mod binar: folosit pentru transferul oricărui tip de fișiere (inclusiv text)

Exemplu de utilizare

Conectarea la un server FTP dintr-o linie de comandă Linux se poate face direct prin comanda **`ftp`**, urmată de numele sau adresa serverului la care se dorește conectarea, ca în exemplul:

```
1 rookie@localhost~$ ftp ftp.lug.ro
2 Trying 193.226.140.51...
3 Connected to ftp.lug.ro (193.226.140.51).
```

```
220 (vsFTPd 2.0.3)
Name (ftp.lug.ro:root): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Promptul `ftp>` indică faptul că următoarele comenzi vor fi trimise direct către server. Câteva comenzi comune sunt ls sau dir pentru a lista conținutul directorului curent, cd pentru a schimba directorul, get pentru a descărca fișierele specificate ca parametri (`mget` pentru fișiere multiple) sau put pentru trimiterea de fișiere. Deconectarea de la serverul FTP se face prin comenzile quit sau disconnect.

O categorie specială de clienți FTP o reprezintă broswerele web, limitarea acestora constând în faptul că nu permit decât descărcarea fișierelor de pe server. Conectarea la un server FTP printr-un browser web se poate face prin introducerea adresei sale în bara de adrese a browserului, specificându-se protocolul:

```
ftp://ftp.lug.ro
```

În cazul serverelor în care accesul nu se poate face anonim (fără cont predefinit), browserul va cere un nume de utilizator și o parolă.

Pentru a specifica direct din adresa parametrii de autentificare, se poate scrie:

```
ftp://user:parola@server:2555
```

Exemplul anterior specifică explicit și portul pe care se realizează conexiunea (2555), informație necesară în cazul în care serverul nu rulează pe portul implicit, 21.

Cuvinte cheie

- client
- curl
- FTP
- HTML
- HTTP
- IMAP
- IP
- LDA
- POP3
- port
- punct-la-punct
- scp
- server
- SMB
- spam
- SSH (Secure Shell)
- stivă de protocoale
- TCP
- telnet
- UDP
- URL
- webmail
- wget
- WWW
- XHTML

Întrebări

1. Care dintre operațiile de mai jos NU se bazează pe o arhitectură client-server?
   - ❑ ridicarea mesajelor
   - ❑ editarea unui nou mesaj
   - ❑ trimiterea unui nou mesaj
   - ❑ verificarea mesajelor folosind broswer
2. Protocolul telnet comunică direct (fără intermedierea nivelului transport) cu nivelul rețea. După stabilirea unei conexiuni de telnet comunicația se va face doar unidirecțional: de la server către client.
   - ❑ adevărat, adevărat
   - ❑ adevărat, fals
   - ❑ fals, adevărat
   - ❑ fals, fals
3. Care dintre serviciile de mai jos NU este util intr-o soluție de webmail?
   - ❑ SMTP
   - ❑ IMAP
   - ❑ HTTP
   - ❑ SSH
4. Care dintre cele de mai jos NU reprezintă o tehnologie relevantă pentru serviciul web:
   - ❑ URL
   - ❑ HTTP
   - ❑ HTML
   - ❑ FTP
5. Care dintre următoarele este o aplicație server SMTP?
   - ❑ Postfix
   - ❑ Firefox
   - ❑ Outlook
   - ❑ Apache
6. Într-un client web (broswer) este tastată adresa:

```
1 ftp://bestman:none@test.com/lista/admisi.txt
```

   Care dintre următoarele afirmații este adevărată?
   - ❑ va fi inițiată o conexiune pe portul 80 către serverul test.com
   - ❑ va fi descărcat fișierul /test.com/lista/admisi.txt

- ❑ se va folosi pentru autentificare parola "bestman"
- ❑ pentru stabilirea conexiunii utilizatorul "bestman" să trebuie să existe pe test.com

7. Pentru conectarea la un server SSH este necesară cunoașterea adresei IP sau a numelui DNS a serverului. Filtrarea mesajelor spam se poate face numai pe serverul de mail, nu și pe client
   - ❑ adevărat, adevărat
   - ❑ adevărat, fals
   - ❑ fals, adevărat
   - ❑ fals, fals
8. Care din următoarele NU este un client web?
   - ❑ Firefox
   - ❑ Internet Explorer
   - ❑ wget
   - ❑ Thunderbird
9. Care dintre următoarele aplicații NU poate fi folosită pentru conectarea la un server web:
   - ❑ telnet
   - ❑ ssh
   - ❑ wget
   - ❑ curl
10. Comunicația între două servere de mail se realizează folosind:
    - ❑ POP3
    - ❑ HTTP
    - ❑ SMB
    - ❑ niciuna dintre variante

Capitolul 10

Elemente de securitate

There are two kinds of cryptography in this world: cryptography that will stop your kid sister from reading your files, and cryptography that will stop major governments from reading your files.

Bruce Schneier

Ce se învață din acest capitol?

- Problematica securității IT
- Elementele unei politici de securitate
- Securizarea sistemului
- Controlul accesului
- Securitatea sistemului de fișiere
- Forme de atac de rețea
- Criptare și firewall-uri
- Monitorizare

10.1 Problematica securității

Securitatea la nivelul calculatoarelor, în particular, și a domeniului IT, în general, a căpătat un interes deosebit în ultimele decenii. Deși un concept discutat din cele mai vechi timpuri, securitatea a căpătat noi forme, ramuri de proiectare și implementare odată cu dezvoltarea sistemelor de calcul și îndeosebi a Internetului. Pe o planetă în care peste 1.5 miliarde de oameni sunt conectați la Internet (circa 25% din populație)[1] securitatea și integritatea informației au o relevanță deosebită. În domeniul IT, domeniul securității a cunoscut o dezvoltare alertă, cu multe subdomenii și specialiști. Se discută despre ingineria securității, despre criptografie, gestiunea riscului, securitatea rețelei, inginerie socială, viruși și antiviruși, scrierea de cod sigur (secure coding).

[1] http://www.internetworldstats.com/stats.htm

În ciuda faptului că domeniul este unul vast, nivelul de cunoaștere și înțelegere a publicului larg în privința securității este destul de scăzut. De-a lungul ultimilor ani, multe teme tehnice specifice domeniului securității au părăsit domeniul IT, fiind preluate de ziare, jurnale TV, sau de industria cinematografică. Din păcate procesul nu a fost, cel mai adesea, unul ce încerca aducerea în sfera publică a conceptelor de securitate, ci mai ales unul ce speculează senzaționalul prin ignorarea constrângerilor lumii reale, ducând la promovarea unor noi mituri ale erei IT.

De multe ori se consideră normale "performanțele" hackerilor din filme, care reușesc să compromită securitatea unui sistem în câteva secunde. Exemple de "scenarii" care ajută la răspândirea acestui fenomen sunt scrierea de viruși pentru sisteme de operare extraterestre (vezi "Ziua Independenței", 1997) sau folosirea unui ecran 3D pentru virusarea unui sistem (vezi "Swordfish", 2001). Există, însă, și exemple pozitive în lumea cinematografică, precum folosirea **`nmap`** și SSH pentru compromiterea unei rețele electrice computerizate (vezi "Matrix", 1997[1]).

Capitolul de față nu își propune să ofere cititorului sfaturi despre obținerea unei puteri nemărginite în controlul tuturor sistemelor electronice, ci să aducă un nivel minim de ordine în domeniul populat de mituri al securității IT. A deveni cu adevărat un profesionist în securitate este o carieră, după cum spunea și Eric S. Raymond: *"Being able to break security doesn't make you a hacker anymore than being able to hotwire cars makes you an automotive engineer."*

10.1.1 Principii de bază

Securitatea calculatoare are drept scop protejarea informațiilor de la accese neautorizate, furt, corupere și alte riscuri. Domeniile securității se referă atât la mecanismele folosite pentru a obține accesul sau a corupe informațiile cât și la protejarea acestora. Forma generală de risc în securitatea IT se numește atac de securitate.

O primă clasificare a riscurilor de securitate distinge trei tipuri de atacuri:

- atacuri venite din Internet (cu o rată de succes redusă);
- atacuri inițiate din rețeaua locală;
- atacuri generate de pe aceeași mașină, acestea din urmă având un impact mult mai însemnat decât primele.

Deși cu gradul de risc cel mai ridicat, atacurile inițiate de utilizatorii serverului țintă sunt deseori tratate în grabă și unitar.

Domeniul securității dispune de un set de principii generale. Acestea nu sunt aplicabile doar în cadrul domeniului IT, ci în domeniul securității în general. Aceste principii nu sunt exhaustive, dar urmărirea acestora facilitează proiectarea și implementarea unui sistem cu nivel de siguranță cât mai ridicat.

Privit din perspectiva unui sistem IT, o soluție de securitate trebuie să includă atât o politică de securitate, ce definește drepturile și responsabilitățile utilizatorilor, cât și

[1] http://nmap.org/images/matrix/

specificații ale asigurării securității fizice, ale componentelor sistemului de operare, ale aplicațiilor locale, precum și ale serviciilor de rețea.

Unul dintre principiile fundamentale ale securității, fie ea IT sau de orice altă natură, este: **securitatea unui sistem este egală cu securitatea celei mai slabe verigi**.

Altfel spus, degeaba îți pui ușă ultra-performantă dacă nu folosești cheia, sau aplicat în domeniul IT: nu are rost să cheltui sume enorme de bani pe sisteme de securitate, dacă utilizatorii folosesc drept parole propriul nume, sau își țin parola lipită pe monitor.

În orice sistem există entități active (utilizatori, procese) și drepturi pe care acestea le au asupra sistemului. În condițiile asigurării securității sistemului, fundamental este **principiul celui mai mic privilegiu**. Acest principiu impune alocarea drepturilor minime entităților active: doar ceea ce este nevoie.

Un administrator de sistem va avea acces la întreg sistemul și resursele acestuia. Un utilizator obișnuit va avea acces doar la acele resurse care îi sunt necesare. Câteva de mecanisme care respectă acest principiu sunt:

- folosirea unui director de tip home cu drepturi complete pentru fiecare utilizator;
- folosirea comenzii **sudo** și fișierului `/etc/sudoers` pentru a conferi anumite privilegii utilizatorilor; (vezi secțiunea 10.3.3)
- separația între informațiile din fișierul `/etc/passwd` și `/etc/shadow`; (vezi secțiunea 10.2.3)
- folosirea **doar** a dreptului de execuție pe directoare când se dorește parcurgerea acestora; (vezi secțiunea 10.2.4)

Acest principiu este legat de **principiul limitării drepturilor** (vezi secțiunea 10.3.3). Întrucât resursele sistemului sunt limitate, un utilizator poate ajunge să folosească o foarte mare parte a acestora în dăuna altor utilizatori sau a sistemului de operare, putând conduce ușor la suspendarea funcționării acestuia. În general, limitarea drepturilor previne atacuri de tipul **DoS**[1] (*denial-of-service*).

Un utilizator poate crea foarte multe procese ducând la ocuparea memoriei (*fork bomb*[2]), un atac de rețea poate deschide multe conexiuni[3] care ocupă memorie, un utilizator poate ocupa spațiu pe disc și împiedica folosirea acestuia. Soluția este configurarea sistemului să limiteze drepturile utilizatorilor sau proceselor sistemului pentru a preveni suprautilizarea resurselor sistemului. Spre exemplu, folosirea cotelor[4] previne supraîncărcarea spațiului de pe disc.

Asocierea dintre o entitate activă (denumită și **subiect**) și drepturile pe care aceasta le deține asupra unor resurse (denumite și **obiecte**) se numește **controlul accesului** (*access control*). Sistemele de operare folosesc diferite mecanisme pentru a stabili când, cât de mult, cum și dacă poate un proces (subiect) să folosească o resursă (obiect).

O politică de securitate trebuie să stabilească un **compromis între gradul de flexibilitate a serviciilor IT și nivelul de securitate dorit**. Luate ad literam, cerințele de securitate ar presupune izolarea totală a sistemului de lumea exterioară, dar cum o

[1] http://en.wikipedia.org/wiki/Denial_of_service
[2] http://en.wikipedia.org/wiki/Fork_bomb
[3] http://en.wikipedia.org/wiki/SYN_flood
[4] http://en.wikipedia.org/wiki/Disk_quota

astfel de abordare duce la limitarea funcționalității, cel mai adesea securitatea unui sistem este definită ca un set de metode de protecție menite să descurajeze și să întârzie atacatorul.

Asigurarea flexibilității unui sistem presupune adăugarea de noi funcționalități. Aceasta se traduce, în general, în crearea unei aplicații complexe. Se spune "complexitatea este dușmanul securității"[1]. Un principiu de bază pentru asigurarea securității este **simplitatea**[2]. Orice nouă caracteristică adăugată unui sistem sau unui program poate însemna introducerea unei noi vulnerabilități. Se estimează că într-o companie specializată pentru dezvoltarea de programe, se introduc zeci de defecte (*bug-uri*) la fiecare 1000 de linii de cod. În dezvoltarea software, opusul simplității se numește *feature creep*[3]: adăugarea de funcționalități care nu sunt necesare.

Un sistem cu nivel foarte bun de securitate se poate obține mai ușor dacă planificarea acestuia ține cont de securitate decât dacă mecanismele de securitate sunt implementate după o perioadă în care acesta a fost dezvoltat. Planificarea unui sistem ținând cont de securitate (*design with security in mind*) este un factor decisiv pentru obținerea unui sistem cât mai sigur.

Dacă în proiectarea unei aplicații, a unei rețele sau a unui sistem, se ține cont de posibilile atacuri, de principiile de bază ale securității, de mediul și contextul de utilizare a aplicației, probabililitatea obținerii unui sistem vulnerabil este scăzută. De asemenea, orice noi mecanisme de asigurare a unui nivel suplimentar de securitate vor fi mai ușor adăugate sistemului. Ingineria sofware folosește denumirea de *secure by design*[4]. Un exemplu este sistemul de operare OpenBSD[5], care pune accent pe obținerea unui sistem cât mai sigur sigur.

Securitatea unui sistem nu este o finalitate, ci un proces. Un sistem nu va fi niciodată perfect sigur, cât timp acesta este folosit. Securitatea unui sistem presupune un șir de acțiuni continue pentru a preveni posibile atacuri. În general, securitatea se traduce în resursele pe care le poate investi un atacator față de resursele pe care le investește cel care protejează sistemul (bani, timp, personal).

În cartea "Secure Coding: Principles and Practices" [8] (capitolul 2 "Architecture"), se menționează că, într-un interviu de angajare, răspunsul potrivit la întrebarea "Cât de sigură poți să faci aplicația mea?" este "Cât de sigură vrei să fie?". Răspunsurile "Pot să o fac sigură în fața oricărui atac." sau "Pot să o fac cât de sigură se poate." ar trebui să rezulte în eliminarea candidatului din lista de potențiali angajați. Un sistem poate fi mai sigur decât un alt sistem, dar acest lucru se realizează cu **un număr crescut de resurse, stres și funcționalități obscure**. Întotdeauna trebuie cântărit cât de sigur se dorește a fi un sistem la câte resurse pot fi investite.

Nu se poate obține un sistem perfect sigur, dar un sistem trebuie menținut la un nivel de securitate. Operațiuni periodice de mentenanță, monitorizare, actualizare, verificare sunt necesare pentru asigurarea unui nivel dorit de securitate. O divizie responsabilă cu asigurarea securității unui sistem va trebui să considere în permanență **exteriorul ca fiind o zonă nesigură**. Tot ceea ce nu face parte din sistemul controlat trebuie

[1] http://www.schneier.com/news-038.html
[2] http://en.wikipedia.org/wiki/KISS_principle
[3] http://en.wikipedia.org/wiki/Feature_creep
[4] http://en.wikipedia.org/wiki/Secure_by_design
[5] http://www.openbsd.org/

considerat ca o zonă cu potențiale riscuri sau atacatori. Orice vulnerabilitate a sistemului trebuie reparată la timp pentru a nu permite unui potențial atacator să o folosească.

Depinzând de domeniu, pot exista și alte principii pentru asigurarea securității. Principiile prezentate mai sus reprezintă câteva din principiile de bază ale securității. Aceste principii nu sunt complete și nu sunt aplicate o singură dată. Pentru a obține un sistem cu nivel cât mai bun de securitate, principiile de bază trebuie revizuite, mecanismele folosite trebuie îmbunătățite și, în general, trebuie executate acțiuni periodice de evaluare a securității sistemului.

10.1.2 Termeni

Domeniul securității dispune de o serie de termeni care descriu la nivel general conceptele de bază folosite. Înțelegerea acestor termeni este fundamentală pentru orice specialist în IT și mai ales pentru cei care doresc să urmeze o carieră în domeniul securității.

Mai jos sunt prezentați o parte din termenii de bază ai domeniului:

- **amenințare** (*threat*) se referă la orice eveniment sau circumstanță cu potențial de a produce pagube unui sistem prin access neautorizat, distrugerea sau modificarea datelor sau denial of service; amenințările provin ca urmare a acțiunilor umane sau a cauze naturale;
- **autentificare** (*authentication*) înseamnă verificarea identității unui utilizator, proces sau dispozitiv, în general ca cerință pentru a permite acestuia accesul la resursele sistemului;
- **autorizare** (*authorization*) este oferirea sau respingerea drepturilor de acces pentru un utilizator sau proces;
- **confidențialitate** (*confidentiality*) este scopul securității care solicită protejarea datelor de la încercări intenționate sau accidentale de vizualizare/citire; confidențialitatea se referă atât la datele aflate pe un dispozitive de stocare, cât și la cele aflate în procesare sau în tranzit;
- **control acces** (*access control*) înseamnă permiterea folosirii autorizate a unei resurse, simultan cu prevenirea folosirii neautorizate sau într-un mod neautorizat;
- **integritate** (*integrity*) este scopul securității care solicită protejarea de la încercări intenționate sau accidentale de alterare a integrității datelor; **integritatea datelor** presupune ca acele date să nu fi fost alterate/modificate într-un mod neautorizat; la fel ca și confidențialitatea se referă atât la date aflate pe dispozitive de stocare, cât și la date aflate în procesare sau în tranzit;
- **denial of service** este un tip de atac care are ca efect prevenirea accesului autorizat la o resursă sau întârzierea operațiilor critice în raport cu timpul;
- **identitate** (*identity*) reprezintă o informație unică în cadrul unui domeniu de securitate, recunoscută ca o entitate unică în acel domeniu;
- **risc** (*risk*) reprezintă probabilitatea ca o anumită amenințare să exploateze o vulnerabilitate a sistemului și impactul care ar rezulta în urma acestei acțiuni;

- **vulnerabilitate** (*vulnerability*) se referă la o slăbiciune la nivelul proiectării, implementării sau operării sistemului, care poate fi declanșată intenționat sau accidental, rezultând într-o violare a politicii de securitate a sistemului.

Există o serie de confuzii de termeni de nume pentru persoanele implicate în domeniul securității. De multe ori noțiunea de *hacker* este folosită generic pentru a cataloga atacatorii de rețea deși, mai corect, denumirea corectă este cea de *cracker* (sau *black-hat hacker*). Următoarea listă reprezintă denumirile uzuale folosite și semnificațiile acestora:

- **hacker** este, în sensul său inițial, un programator caracterizat prin curiozitate și dorință de rezolvare a problemelor existente (de unde noțiunea de *hacking*[1]; în sensul securității, noțiunea de hacker poate avea două înțelesuri: *white-hat hacker* (*ethical hacker*) sau *black-hat hacker* (*cracker*);

- **cracker** (sau **black-hat hacker** este specializat în atacatarea unor sisteme pentru obținerea accesului; motivația poate fi financiară, politică, sau pentru distracție; scopul este, în general, distrugerea, furtul sau alterarea informației;

- **script kiddie** este un termen peiorativ folosit pentru a descrie persoanele fără cunoștințe tehnice deosebite pentru a fi considerați **hackeri** dar care folosesc scripturi sau programe scrise de alții pentru a ataca sisteme de calcul și rețele;

- **ethical hacker** (sau **white-hat hacker** este un expert în securitate, specializat în metodologii de testare a vulnerabilității unui sistem sau a unei rețele pentru a asigura un nivel cât mai bun de securitate; în general, ethical hackerii aplică metode similare crackerilor, dar cu scopul final de testare și securizare a sistemului atacat.

10.2 Securizarea sistemului

Securitatea sistemului se referă la mijloacele prin care se poate proteja un sistem de calcul. Atacurile pot veni din exterior (din Internet, sau din rețeaua locală), pot veni din interior (de la utilizatori) sau pot fi cauze naturale (căderea tensiunii poate conduce la pierderea unui hard disk sau la coruperea datelor).

Protejarea sistemului nu se referă doar la protecția sistemului de operare și a aplicațiilor ce rulează peste acesta ci și la protecția fizică a sistemului: poziționarea în zone sigure, folosirea de uși metalice, prevenirea accesului persoanelor neautorizate.

10.2.1 Securitatea sistemului de operare

Securitatea aplicațiilor trebuie să pornească de la asumarea gradului de risc dat de sistemul de operare. Dintre componentele sistemului de operare, o mare parte a atacurilor încearcă să exploateze limitări ale implementărilor de separare a drepturilor de acces la memorie sau la sistemul de fișiere.

[1] http://en.wikipedia.org/wiki/Hacker_(programmer_subculture)

Un sistem de operare este sigur dacă resursele acestuia (zone de memorie, dispozitive de intrare/ieșire, fișiere etc.) sunt accesate în mod valid de entitățile active (în general procese). Accesul valid este asigurat de nucleul sistemului de operare, ce acționează ca un intermediar între utilizatori și componentele hardware.

Pornind de la sistemele de operare simple (cele pentru PDA) până la sistemele complexe ce rulează pe calculatoarele personale, securitatea oferită de nucleu se bazează pe suport hardware. Procesoarele oferă cel puțin două niveluri de privilegii:

- un nivel de privilegii pentru operații obișnuite;
- un altul pentru accesul la instrucțiuni privilegiate: accesul la zone de memorie rezervată; accesul la dispozitivele hardware etc.

Accesul la aceste instrucțiuni privilegiate este permis doar nucleului.

Securitatea memoriei

În sistemele de operare moderne, securitatea memoriei este strâns legată de mecanismul de memorie virtuală. Fără a intra în detalii, este suficient de precizat faptul că sistemele de operare asigură fiecărui proces un spațiu dedicat de memorie virtuală (denumit și spațiu de adrese) care este mapat/asociat unor zone din memoria RAM. Zonele de memorie RAM peste care este mapat spațiul de adrese al unui proces nu poate fi accesat de către alte procese.

Problemele de securitate la nivelul memoriei țin de detectarea corectă a încercărilor de accesare a unor zone de memorie care nu se află în spațiul de adresă al procesului respectiv, dar și a încercărilor de schimbare a unor zone de memorie în spațiul propriu de adrese, pe parcursul rulării procesului.

Fiecare proces are propriul său spațiu de memorie virtuală mapat peste un spațiu de memorie fizică printr-o tabelă de translatare ce aparține procesului. Pentru un proces care încearcă să acceseze o adresă de memorie pentru care nu există o translatare către o adresă fizică (adică pentru care nu există pagina respectivă de memorie în RAM), procesorul va verifica la nivel hardware dacă respectiva adresă se află în spațiul de adrese al procesului. Dacă se află, înseamnă că respectiva pagină de memorie a fost evacuată pe disc (swap) și trebuie adusă în RAM. În caz contrar va fi generat un semnal de pagină (de memorie) invalidă (denumit page fault[1]), pe baza căruia nucleul va putea decide suspendarea sau terminarea procesului în cauză. În sistemele Unix procesul primește semnalul `SIGSEGV`, și are ca efect terminarea procesului cu mesajul *"Segmentation fault"*.

Toate aceste operațiuni cad în sarcina procesorului și sunt realizate la nivel hardware.

Protecția memoriei este o componentă esențială a sistemelor multitasking (sisteme ce oferă posibilitatea mai multor procese de a se afla în stare de execuție în același timp). Din punct de vedere hardware, primele procesoare pentru calculatoarele personale ce oferă suport pentru protecția memoriei sunt procesoarele din familia Intel 80386[2]. Folosind mecanisme împrumumate de la DOS, care rula pe sisteme fără suport

[1] http://en.wikipedia.org/wiki/Page_fault
[2] http://en.wikipedia.org/wiki/Intel_80386

hardware (8086, 80286), versiunile de Windows până la Windows 98 și Windows ME nu au oferit o protecție completă a memoriei.

10.2.2 Controlul accesului

Controlul accesului se referă la oferirea accesului autorizat și prevenirea acccesului neautorizat la sistem. Poate fi vorba de accesul fizic al unei persoane sau de accesul unui utilizator în cadrul sistemului de operare.

Securitatea fizică

Securitatea fizică include o gamă largă de parametri, de la controlul accesului personal până la asigurarea diferiților parametri optimi de funcționare.

O primă componentă a securității fizice presupune plasarea echipamentelor în locuri în care se poate asigura controlul accesului, pentru a preveni utilizarea neautorizată, distrugerea sau furtul.

Specificațiile de alimentare cu tensiune vor include evaluări ale consumului de putere pentru dimensionarea surselor neinteruptibile de putere (UPS-uri), sau a generatoarelor de tensiune.

Asigurarea temperaturii optime în centrele de operații ține de folosirea unor sisteme de aer condiționat sau de încălzire cu un grad ridicat de redundanță.

Există și cerințe de protecție împotriva incendiilor: o cameră cu servere va trebui să aibă, pe lângă sisteme de detecție și stingere a incendiilor, și podea și pereți din materiale rezistente la foc.

O bună politică de securizare va urmări plasarea echipamentelor în bunkere aflate la subsolul clădirilor, urmărindu-se astfel și protecția împotriva cutremurelor.

Securitatea la nivelul utilizatorilor

Una dintre componentele esențiale ale unei politici de securitate este securitatea la nivelul utilizatorilor. Securitatea utilizatorilor se bazează pe separarea resurselor fiecărui utilizator, precum și pe folosirea autentificării pentru protejarea accesului la resurse.

Principiul important de prevenire a atacurilor la nivelul utilizatorilor este principiul celui mai mic privilegiu. Orice operațiune trebuie executată dintr-un cont cu drepturi cât mai limitate.

Un administrator novice va folosi contul privilegiat inclusiv pentru efectuarea unor operațiuni uzuale. Exploatarea unor ambiguități ale variabilelor de mediu poate duce în acest caz la rularea accidentală a unei aplicații malițioase lăsată într-un spațiu public (gen `/tmp/`), rulare ce nu ar fi fost posibilă dintr-un cont de utilizator. Una dintre criticile aduse sistemelor din familia Windows a fost crearea unui cont implicit de tip Administrator care oferă drepturi depline utilizatorului sistemului, inclusiv instalarea

unor programe malițioase. Începând cu Windows Vista, acest lucru a fost combătut prin folosirea User Account Control[1].

10.2.3 Parole

Deși mecanismele de autentificare s-au diversificat, în continuare o pondere semnificativă o constituie perechile `<utilizator, parolă>`. O soluție alternativă, ce oferă un grad mult mai ridicat de scalabilitate, precum și de securitate, este folosirea de certificate. Conceptul ce stă la baza folosirii de certificate se numește PKI[2] (*Public Key Infrastructure*).

Numărul resurselor electronice fiind în continuă creștere, mulți utilizatori ajung să comită cel puțin una dintre erorile de căpătâi ale autentificării: alegerea de parole ușor de ghicit (spart), folosirea unei parole pentru mai multe conturi sau notarea parolelor în locuri nesigure.

Alegerea parolelor

Una dintre principalele probleme în alegerea parolelor este folosirea de parole prea simple. Mulți utilizatori folosesc numele propriu, numele soției, copiilor sau câinelui, o dată de naștere, numele echipei preferate. Aceste parole pot fi ușor ghicite printr-un atac de tipul dicționar.

Se spune că, o parolă eficientă:

- trebuie să aibă un număr de minim 7-8 caractere;
- trebuie să folosească atât minuscule, cât și majuscule și cifre;
- trebuie să includă cel puțin un caracter special (și nu doar pe ultima poziție în cadrul parolei);
- nu trebuie să fie un cuvânt din dicționar;
- nu trebuie să fie un nume de persoană.

Cu toate acestea, **parolele trebuie să fie ușor de reținut**. În această situație, cea mai la îndemână soluție este alegerea unei fraze pe baza căreia se poate deduce parola (vezi Ross Anderson – Security Engineering, secțiunea 3.3.3 [3]). *"Come to the dark side, we have cookies!"* poate fi folosită pentru a reține parola *"Cttds, whc!"*, după cum *"Social engineering bypasses all technologies, including firewalls"* poate genera *"S3bat,1f"*.

În cazul folosirii unei parole ce nu respectă recomandările de mai sus, atacurile bazate pe dicționar (încercarea parolelor ce aparțin unui set de cuvinte comune), sau cele bazate pe forță burtă (încercarea tuturor combinațiilor de caractere) pot avea rezultate aproape imediate. Există numeroase aplicații ce simulează astfel de atacuri, una dintre cele mai renumite fiind **John the Ripper**[3].

---

[1] http://en.wikipedia.org/wiki/User_Account_Control
[2] http://en.wikipedia.org/wiki/Public_key_infrastructure
[3] http://openwall.com/john/

De la conturi triviale de download până la conturile de email, sau la conturile bancare electronice, tot mai multe resurse solicită o autentificare pe bază de parolă. Este ideală **separarea conturilor prin parole individuale**. Cum acest lucru presupune memorarea unui număr ridicat de fraze, o soluție de compromis constă în folosirea unor parole comune pentru servicii similare cu miză de securitate redusă. Astfel, nu este prudentă folosirea PIN-ului cardului de bancă pentru contul de mail; un atac de forță brută ar dura aproximativ o microsecundă. Se poate însă folosi aceeași parolă pentru mai multe site-uri de descărcare de software.

Deși folosirea unor parole comune pentru servicii electronice cu miză mică reduce numărul parolelor, acestea pot fi însă, destul de multe. În plus, politicile de securitate solicită schimbarea periodică a parolelor: timpul de viață recomandat pentru o parolă este de șase luni. În măsura în care parolele folosite nu pot fi memorate, vor trebui notate. În mod evident, notarea parolelor nu trebuie făcută pe o foaie cu acces public. O soluție pentru parolele rar folosite constă în păstrarea lor centralizată. Există numeroase aplicații menite să protejeze printr-o singură parolă o bază de date de parole (spre exemplu **KeePass**[1].

Conștientizarea importanței unei parole nu ține doar de protejarea unor resurse personale, ci de gradul de securitate a întregului sistem, precum și a rețelei locale. Altfel spus, într-o rețea în care utilizatorii folosesc parole triviale, riscul nu este doar unul personal. Un atacator poate să compromită din Internet un cont local, transformând un atac de la distanță într-unul generat de pe aceeași mașină, ceea ce îi crește mult șansele de succes. Acesta este și motivul pentru care au evoluat și mecanismele prin care administratorii de rețea pot forța folosirea unor parole eficiente.

Atât rețelele Windows, cât și cele Linux oferă suport pentru securitatea bazată pe **jetoane hardware** (*security token*[2]) ce își regenerează parola la fiecare 30 sau 60 de secunde. Odată sincronizat ceasul serverului cu ceasul jetonului hardware, utilizatorul va folosi numele său de cont și parola afișată de jeton la momentul autentificării. Prețul unei astfel de soluții este de 5-10 euro per jeton hardware, fiecare jeton trebuind reconfigurat la 6 luni. Trebuie remarcat totuși că această soluție nu rezolvă riscurile de securitate fizică a parolei.

Gestiunea parolelor în Unix

Parolele erau, inițial, stocate în fișierul `/etc/passwd`, dar, pe măsură ce precalcularea și stocarea parolelor comune[3] a devenit posibilă, a apărut nevoia unui fișier separat, cu permisiuni mai stricte, în care sunt stocate parolele. Acesta se numește `/etc/shadow` și formatul său, cât și al `/etc/passwd`, este descris în secțiunea 3.3.1.

Utilitarul pentru schimbarea parolei în Unix este **`passwd`**. Utilizatorul privilegiat (root) poate schimba parola oricărui utilizator, inclusiv a sa. Un utilizator neprivilegiat poate schimba doar parola proprie. Înainte de a schimba parola îi este solicitată parola proprie pentru a preveni situația în care un utilizator uită o sesiune shell deschisă.

```
1 alina@valhalla:~$ passwd
2 Changing password for alina.
```

[1] http://keepass.info/

[2] http://en.wikipedia.org/wiki/Security_token

[3] Un astfel de atac se numește *dictionary attack*: http://en.wikipedia.org/wiki/Dictionary_attack

```
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Dacă se dorește obținerea unei parole ușor de reținut, dar care să fie relativ sigură, se poate folosi utilitarul **pwgen**. **pwgen** generează în mod implicit parole de 8 caractere (incluzând cifre):

```
alina@valhalla:~$ pwgen -N 1
uo3Pheis

alina@valhalla:~$ pwgen -N 2
Eeb7Aej9 Xoo0beir
```

Utilitarul permite "personalizarea" parolelor obținute. În exemplul de mai jos, parametrul `-n` precizează faptul că în parolă va fi inclusă cel puțin o cifră, parametru `-c` de va defini lungimea parolei dorite. Ultima cifră reprezintă numărul de parole ce trebuie generate.

```
alina@valhalla:~$ pwgen -n -c 7 3
ahngeZ5 aiv5Iec bie3poR
```

10.2.4 Securitatea sistemului de fișiere

Securitatea sistemului de fișiere urmărește asigurarea izolării utilizatorilor prin separarea fișierelor personale și prin definirea acțiunilor permise de utilizatori pe diverse secțiuni din ierarhia sistemului de fișiere.

Fiecare utilizator dispune de o intrare (un director) în sistemul de fișiere pentru care are drepturi depline. În Unix, acest director se găsește în `/home`. În Windows Vista, acest director este subdirector al `C:\Users`. Utilizatorul poate controla acțiunile permise pe fișierele conținute în respectivul director. Unele intrări în sistemul de fișiere nu pot fi folosite decât de utilizatorul privilegiat.

Controlul accesului

Controlul accesului la diverse elemente ale sistemului de fișiere diferă între sistemele Windows și cele Unix.

În Windows, sistemul de fișiere NTFS oferă de la versiunea 4.0 posibilitatea definirii unei liste de acces pentru fiecare utilizator din sistem. Această listă de acces este definită în raport cu șapte tipuri de acțiuni: *Full Control*, *Modify*, *Read & Execute*, *List*, *Read*, *Write*, *Special Permisions*. Subiectul drepturilor de acces pe un sistem de fișiere NTFS este tratat mai pe larg în studiul de caz al acestui capitol în secțiunea 10.6.1. Abordarea în cazul Windows este o formă a mecanismului de control a accesului[1].

Abordarea folosită în Unix pentru definirea drepturilor de acces la sistemul de fișiere este o matrice de drepturi.

Pentru a obține o matrice limitată, se folosesc trei trepte de privilegiu:

[1] http://en.wikipedia.org/wiki/Access_control_list

Tabelul 10.1: Matricea drepturilor de acces în Unix

	citire (r)	**scriere (w)**	**execuție (x)**
utilizator			
grup			
alții			

- **utilizatorul** (`user`) care deține fișierul;
- **grupul** (`group`) care deține fișierul;
- **alți utilizatori** (`others`).

Pentru fiecare dintre cele trei trepte de privilegiu sunt definite câte trei drepturi:

- **citire** (`read`) – permite vizualizarea conținutului unui director sau a unui fișier;
- **scriere** (`read`) – permite alterarea conținutului unui fișier sau director. Pentru un director, alterarea conținutului înseamnă posibilitatea ștergerii sau creării de fișiere;
- **execuție** (`execute`) – permite execuția unui fișier și permite parcurgerea unui director; dacă un director nu are drept de execuție, nu poate fi parcurs (nu se poate ajunge la fișierele și subdirectoarele conținute).

Având trei trepte de privilegiu și trei drepturi pentru fiecare treaptă, rezultă un total de 9 drepturi care pot fi configurate pentru un fișier dat. Forma liniarizată a matricei de acces (și a celor 9 drepturi) este afișată de comandă **`ls`** cu opțiunea de *long listing* (`-l`):

```
1 root@rosedu:~:~# ls -ld /var/svn/hfall/db/
2 drwxr-xr-x 6 hfall projects 4096 Jul 23 13:34 /var/svn/hfall/db/
```

În listing-ul de mai sus, utilizatorul asociat directorului `/var/svn/hfall/db/`, (`hfall`), are drepturi complete (`rwx`), iar grupul asociat (`projects`) și ceilalți utilizatori au drept de citire și execuție (`r-x`) (se poate vizualiza și parcurge directorul).

Forma liniarizată a matricei de acces pentru un fișier (afișată de comanda **`ls -l`**) este prezentată în figura 10.1.

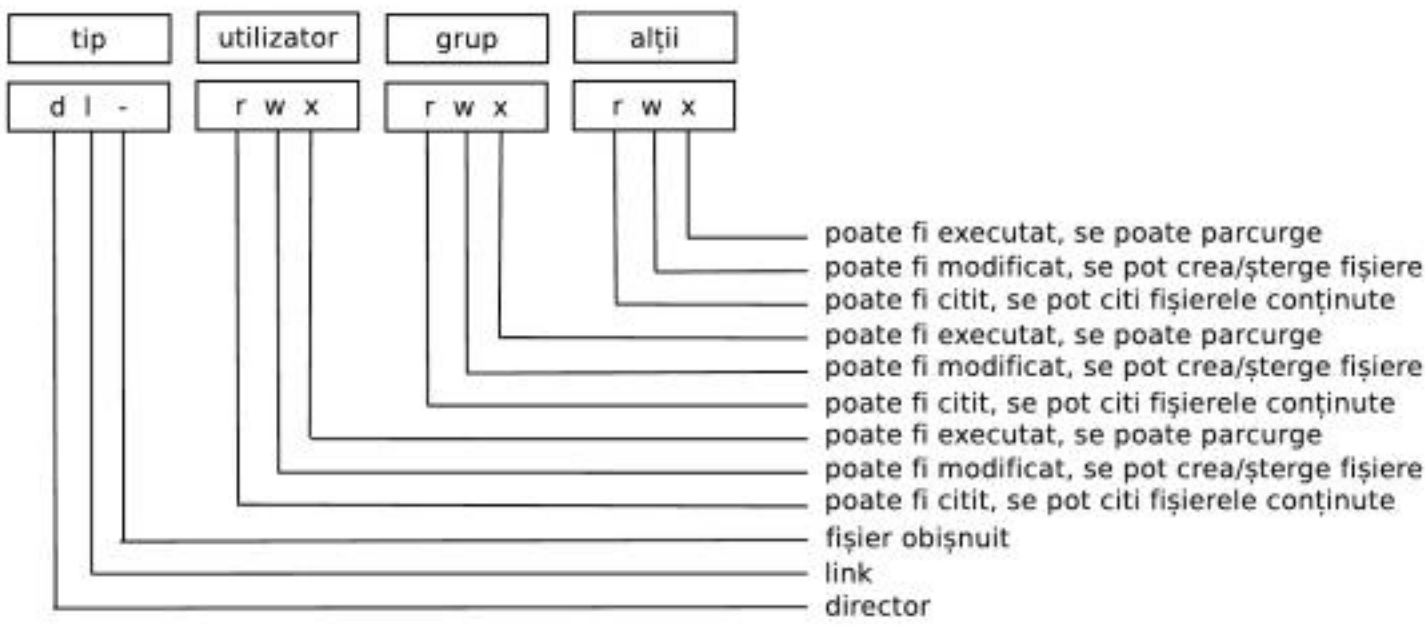


Figura 10.1: Permisiuni Unix

Pentru a specifica mai succint drepturile de acces la un fișier, utilizatorii avansați Unix preferă **forma octală a drepturilor de acces** la un fișier. În cadrul acestei forme, fiecare drept de acces este înlocuit cu un bit. Astfel, în cazul listing-ului de mai sus, drepturile de acces sunt:

```
1 rwx r-x r-x
2 111 101 101
```

Fiecare pereche de trei biți (asociată unei trepte de privilegiu) este tranformată într-o valoare octală. Forma finală, în octal, pentru drepturile de mai sus este, așadar `755`. În tabelul 10.2 sunt prezentate câteva exemple de asociere între forma literală și forma octală a dreputurilor de acces pe fișiere în Unix.

Tabelul 10.2: Permisiuni în format literal și octal

Format literal	Format binar	Format octal
r-xr-----	101 100 000	540
rw--w--wx	110 010 011	623
rwxr-xrw-	111 101 110	756
rwx--x--x	111 001 001	711
rw-r--r--	110 100 100	644

chown și chmod

În administrarea securității sistemului de fișiere în Unix, utilitarele de bază sunt **`chown`** și **`chmod`**. Primul oferă posibilitatea schimbării proprietarului și a grupului căruia îi aparține respectivul fișier, în vreme ce al doilea permite modificarea drepturilor.

```
1 root@kiwi:~# chown -R george:users test/
```

În exemplul de mai sus a fost efectuată o schimbare recursivă pentru toate fișierele și subdirectoarele directorului `test/`. Întreg conținutul directorului `test/` va aparține utilizatorului `george` și grupului `users`.

Dacă se dorește doar modificarea utilizatorului sau grupului, se folosește doar o parte a sintaxei:

```
1  razvan@valhalla:/tmp$ chown alina battleship
2  chown: changing ownership of 'battleship': Operation not permitted
3
4  razvan@valhalla:/tmp$ sudo bash
5
6  root@valhalla:/tmp# chown alina battleship
7
8  root@valhalla:/tmp# ls -l battleship
9  -rw-r--r-- 1 alina razvan 0 Sep 19 18:01 battleship
10
11 root@valhalla:/tmp# chown :shadow battleship
12
13 root@valhalla:/tmp# ls -l battleship
14 -rw-r--r-- 1 alina shadow 0 Sep 19 18:01 battleship
```

În exemplul de mai sus a fost schimbat utilizatorul și grupul fișierul `/tmp/battleship` în `alina`, respectiv `shadow`. Se observă că în Linux, un utilizator neprivilegiat nu

poate schimba deținătorul unui fișier. În Linux, comanda **chown** este folosită **doar** de către utilizatorul privilegiat. Pe alte Unixuri (spre exemplu, Solaris) comanda poate fi folosită și de un utilizator neprivilegiat.

Comanda **chmod** permite modificarea drepturilor de acces ale unui fișier. Comanda poate fi folosită doar de utilizatorul ce deține fișierul sau de utilizatorul privilegiat. Noile drepturi ale fișierului pot fi precizate în formă literală sau octală.

Fie fișierul `hello.c` de mai jos:

```
root@kiwi:~# ls -l
hello.c -rw-r--r--  1 razvan new 81 Oct  6 21:35 hello.c
```

Pentru schimbarea drepturilor se va folosi mai întâi forma literală. Drepturile pot fi precizate pentru oricare dintre cele trei niveluri de privilegiu: utilizator, grup, alți utilizatori printr-o singura literă: **u**, **g**, **o**. Drepturile pot fi:

- adăugate prin folosirea operatorului +;
- înlăturate prin folosirea operatorului −;
- precizate explicit prin folosirea operatorului =.

```
root@kiwi:~# chmod o+x hello.c

root@kiwi:~# ls -l hello.c
-rw-r--r-x  1 razvan new 81 Oct  6 21:35 hello.c

root@kiwi:~# chmod u=rx hello.c


root@kiwi:~# ls -l hello.c
-r-xr--r-x  1 razvan new 81 Oct  6 21:35 hello.c
```

În exemplul de mai sus a fost mai întâi adăugat pentru restul utilizatorilor dreptul de execuție, iar la al doilea pas s-a precizat explicit că drepturile utilizatorului trebuie să fie `rx` adică doar citire și execuție.

În format octal, pentru a acorda toate drepturile utilizatorului (`rwx`), drepturi de citire și execuție grupului (`r-x`), și doar drepturi de citire pentru restul utilizatorilor (`r--`) este suficientă o singură comandă:

```
root@kiwi:~# chmod 754 hello.c

root@kiwi:~# ls -l hello.c
-rwxr-xr--  1 razvan new 81 Oct  6 21:35 hello.c
```

umask

Pentru controlul drepturilor unei noi intrări în sistemul de fișiere se folosește un parametru de restricție numit **mască de creare** (*file mode creation mask*). Drepturile efective ce vor fi atribuite unei noi intrări în sistemul de fișiere se obțin prin efectuarea operației de ȘI logic între valoarea inversată a măștii și permisiunile implicite (`666` pentru fișiere și `777` pentru directoare). Reprezentată matematic, operația este `default_perm & ~mask`.

Comanda ce permite inspectarea și modificarea măștii se numește **umask**.

```
root@kiwi:~# umask
0022

root@kiwi:~# touch uso7_test1

root@kiwi:~# ls -l uso7_test1
-rw-r--r--  1 razvan razvan 0 Nov 10 17:28 uso7_test1

root@kiwi:~# umask 027

root@kiwi:~# touch uso7_test2

ro0t@kiwi:~# ls -l uso7_test2
-rw-r-----  1 razvan razvan 0 Nov 10 17:28 uso7_test2

root@kiwi:~# mkdir uso7_test_dir

root@kiwi:~# ls -ld uso7_test_dir/
drwxr-x---  2 razvan razvan 1024 Nov 10 17:29 uso7_test_dir
```

În listingul de mai sus, în primă fază masca are valoarea `0022`. Acest lucru înseamnă că un fișier nou creat (`uso7_test1` va avea drepturile

```
666 & ~022 = 666 & 755 = 110 110 110 & 111 101 101 =
        = 110 100 100 = 644 = rw- r-- r--
```

În ultimul pas, masca are valoarea `027`. Directorul `uso7_test_dir/` va fi creat cu drepturile

```
777 & ~027 = 777 & 750 = 111 111 111 & 111 101 000 =
        = 111 101 000 = 750 = rwx r-x ---
```

10.3 Întreținerea sistemului

După configurarea drepturilor de acces la nivelul sistemului de operare sau al sistemului de fișiere, unui sistem îi trebuie asigurată mentenanța. Aceasta presupune verificarea periodică a nivelului de securitate a sistemului, a jurnalelor, a acțiunilor utilizatorilor și proceselor. Prevenirea atacurilor este cea mai bună formă de asigurare a securității, iar un administrator de sistem profesionist va inspecta periodic sistemul și va lua măsurilor adecvate atunci când siguranța sistemului este periclitată.

10.3.1 Monitorizarea sistemului

Monitorizarea dinamică a sistemului

Monitorizarea dinamică (în timp real) a sistemului urmărește nivelul de încărcare a diferitelor resurse hardware. Astfel, pentru a determina gradul de încărcare a procesorului se poate folosi comanda **`uptime`**. Pentru a urmări nivelul de folosire a memoriei RAM se poate folosi comanda **`free`**.

Comanda **top** (vezi secțiunea 5.2.4) combină informațiile oferite de comenzile **uptime** și **free**.

```
1  top - 14:05:43 up  8:12,  2 users,  load average: 1.01, 1.02, 1.00
2  Tasks:  61 total,   3 running,  58 sleeping,   0 stopped,   0 zombie
3  Cpu(s): 99.7% us,  0.3% sy,  0.0% ni,  0.0% id,  0.0% wa,  0.0% hi,  0.0%
    si
4  Mem:    256740k total,   254200k used,     2540k free,    12680k buffers
5  Swap:   289128k total,     1444k used,   287684k free,    41356k cached
6
7    PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
8   2490 www-data  25   0 26204  18m  12m R 99.5  7.5 275:26.52 apache
9  14067 root      16   0  2064 1036 1852 R  0.3  0.4   0:00.02 top
10     1 root      16   0  1504  512 1352 S  0.0  0.2   0:00.73 init
11     2 root      34  19     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
12     3 root       5 -10     0    0    0 S  0.0  0.0   0:00.00 events/0
13     4 root       8 -10     0    0    0 S  0.0  0.0   0:00.00 khelper
14     5 root      14 -10     0    0    0 S  0.0  0.0   0:00.00 kacpid
15    31 root       5 -10     0    0    0 S  0.0  0.0   0:03.15 kblockd/0
16    41 root      20   0     0    0    0 S  0.0  0.0   0:00.00 pdflush
```

Prima linie conține (în același format) informațiile oferite de comanda **uptime**: ora curentă, intervalul de timp de la ultima inițializare a sistemului (în cazul exemplului 8 ore și 12 minute), numărul de utilizatori autentificați în sistem și încărcarea medie a sistemului pentru ultimele 1, 5 sau 15 minute. O încărcare supraunitară este echivalentă cu un sistem solicitat.

Liniile 4 și 5 oferă informațiile specifice comenzii **free**. Linia 4 descrie încărcarea memoriei RAM, iar linia 5 indică nivelul de utilizare a partiției de swap. Folosirea unei cantități însemnate de memorie din swap indică faptul că memoria RAM este insuficientă pentru cerințele sistemului.

În exemplul de mai sus se observă că procesorul este supraîncărcat, în vreme ce memoria RAM este bine dimensionată (pentru că memoria swap nu este folosită). Inspectând mai departe procesele cele mai relevante din punctul de vedere al consumului de resurse se observă că procesul apache a petrecut în procesor 275 de minute, adică mai mult de jumătate din timpul scurs de la ultima inițializare a sistemului. Se observă, de asemenea, că, în momentul rulării comenzii, procesul `apache` folosea 99.5% din procesor. Acesta este un comportament neobișnuit. Pentru un sistem bine configurat cele mai active procese vor reuși la nivelul unei luni de zile să ocupe 1-2 minute.

Concluzia analizei dinamice a sistemului de mai sus nu este că resursele hardware sunt insuficiente, ci că unul dintre servicii (mai exact serviciul de web) are un comportament anormal. Prima măsură ar trebui să fie oprirea procesului apache și inspectarea configurațiilor specifice acestui daemon.

Monitorizarea utilizării discului

O altă resursă importantă a unui sistem o reprezintă spațiul liber de pe hard-disk. Pentru aceasta trebuie monitorizat nivelul de ocupare a partițiilor definite pe disc, dar și dimensiunea unor directore și fișiere.

Spațiul liber pentru partițiile discului sistemului se poate vizualiza cu ajutorul comenzii **df**:

```
root@cursuri:/home/courses# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda1             4.6G  3.2G  1.2G  74% /
tmpfs                 126M  4.0K  126M   1% /dev/shm
/dev/hda6              32G   30G  1.2G  97% /home
```

În exemplul de mai sus spațiul pe disc este împărțit în două partiții: una folosită pentru rularea sistemului (montată ca `/`) și o a doua folosită pentru stocarea datelor utilizatorilor (montată ca `/home`). În acest fel, chiar dacă spațiul pe partiția `/home` este epuizat, acest lucru nu va afecta funcționarea sistemului.

În cazul în care se dorește inspectarea dimensiunii unui fișier anume se poate folosi comanda `ls` sau comanda `stat`:

```
root@rosedu:~# ls -lh /var/log/apache2/access.log
-rw-r----- 1 root adm 7.2M Sep 19 20:12 /var/log/apache2/access.log

root@rosedu:~# stat -c %s /var/log/apache2/access.log
7482956
```

Pentru a obține dimensiunea unui director se poate folosi comanda `du`:

```
root@cursuri:/home/courses# du -sh rc
155M    rc
```

Monitorizarea serviciilor de rețea

Pentru monitorizarea serviciilor de rețea disponibile pe stația locală este folosit utilitarul **netstat**.

Apelat fără niciun parametru, **netstat** va afișa o listă cu toate conexiunile active (porturile aflate în starea de listen se consideră inactive). O utilizare frecventă a acestui utilitar urmărește testarea funcționalității anumitor servere (HTTP, FTP etc.). În general, aceste servere așteaptă conexiuni TCP sau UDP pe anumite porturi.

Pentru a obține lista tuturor conexiunilor TCP, se poate folosi:

```
root@rosedu:~# netstat --tcp --all
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address   Foreign Address   State
tcp        0      0 *:ssh           *:*               LISTEN
tcp        0      0 *:smtp          *:*               LISTEN
tcp        0      0 *:git           *:*               LISTEN
[..]
```

Pentru a verifica funcționarea diferitelor servere instalate, care ascultă eventuale cereri de conectare, este utilă comanda:

```
root@rosedu:~# netstat --tcp --listening --numeric-ports
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address    Foreign Address   State
tcp        0      0 0.0.0.0:22      0.0.0.0:*         LISTEN
tcp        0      0 0.0.0.0:25      0.0.0.0:*         LISTEN
tcp        0      0 0.0.0.0:9418    0.0.0.0:*         LISTEN
```

```
7 tcp        0      0 127.0.0.1:3306   0.0.0.0:*         LISTEN
8 [..]
```

Se observă că sistemul mașina locală există servere care ascultă pe portul 22 (SSH), 25 (SMTP) sau 3306 (mysql). Așadar, server-ele SSH, SMTP și mysql sunt funcționale.

Pentru vizualizarea conexiunilor UDP, se poate utiliza parametrul `-u` sau `--udp`

Să presupunem că se descoperă un server care funcționează (se află în starea `LISTEN`) pe portul 22. Pentru oprirea acestui server putem opri procesul corespunzător. Înainte însă, trebuie aflat pid-ul acestui proces (opțiunea `-p` sau `--program`):

```
1 root@rosedu:~# netstat --tcp --listening --program --numeric-ports
2 Active Interne connections (only servers)
3 Proto Recv-Q Send-Q Local Address   Foreign Address  State    PID/Program
   name
4 tcp        0      0 0.0.0.0:80      0.0.0.0:*        LISTEN   5721/apache
5 [..]
6 tcp6       0      0 :::22           :::*             LISTEN   5515/sshd
7 tcp6       0      0 ::1:25          :::*             LISTEN   5505/master
```

Ulterior, se poate trimite un semnal `KILL` către procesul cu numărul `5515`.

```
1 root@rosedu:~# kill -KILL 5515
```

Monitorizarea activității utilizatorilor

Pentru a determina ce utilizatori sunt autentificati la un moment dat în sistem se foloseste comanda **`who`**:

```
1 root@cursuri:~# who
2 so       pts/0        Aug 10 13:50 (anaconda.cs.pub.ro)
3 ot       pts/1        Aug 10 14:05 (kiwi.cs.pub.ro)
```

Din exemplul de mai sus se poate deduce că în sistem sunt autentificați utilizatorii `so` și `ot`. Ambii utilizatori au acces în sistem prin emulatoare de terminal (`pts`), cu alte cuvinte prin intermediul unor sesiuni la distanță. Sesiunea utilizatorului `so` a fost inițiată la ora 13:50 de pe stația `anaconda.cs.pub.ro`, în vreme ce sesiunea utilizatorului `ot` a fost începută la 14:05 de pe sistemul `kiwi.cs.pub.ro`.

Pentru a determina ce utilizatori au folosit sistemul în ultima perioadă, se poate folosi comanda **`last`**:

```
1 root@swarm:~# last -5
2 root     pts/1        141.85.37.227    Sat Sep 19 20:24   still logged in
3 root     pts/1        188.27.105.225   Sat Sep 19 18:21 - 18:39  (00:17)
4 andreif  pts/2        81.181.250.24    Sat Sep 19 14:55 - 14:56  (00:00)
5 andreif  pts/2        81.181.250.24    Sat Sep 19 14:25 - 14:41  (00:15)
6 andreif  pts/2        81.181.250.24    Sat Sep 19 14:01 - 14:25  (00:24)
7
8 wtmp begins Tue Sep  1 08:01:55 2009
```

10.3.2 Jurnalizarea și gestiunea jurnalelor

Majoritatea serviciilor sistemului rulează ca daemoni (vezi secțiunea 5.3) și nu pot comunica direct cu utilizatorul. Utilizatorul interacționează cu serviciile prin intermediul

semnalelor și a fișierelor de configurare, iar serviciile oferă informații de stare, raportare și funcționare prin intermediul fișierelor jurnal (`server logs`).

Fișierele jurnal pot avea diverse formate, dar, în general, fiecare linie de tip jurnal precizează ora și data, adresa IP cu care serviciul a comunicat și o descriere a mesajului. Linia de mai jos este o parte dintr-un jurnal al unui server Apache în *Common Log Format*. În cazul de față este vorba de o conexiune inițiată de web crawler-ul de la Yahoo![1].

```
1 72.30.78.249 - - [19/Sep/2009:20:29:26 +0300] "GET /tag/picasa/ HTTP/1.0"
   200 6331 "-" "Mozilla/5.0 (compatible; Yahoo! Slurp/3.0; http://help.
  yahoo.com/help/us/ysearch/slurp)"
```

Pe un sistem Unix, jurnalele se găsesc, de obicei, în `/var/log/`:

```
1 root@swarm:~# ls /var/log
2 apache2           daemon.log.4.gz  kern.log        mail.log.4      syslog
3 apt               debug            kern.log.1      mail.log.5.gz   syslog
  .0
4 auth.log          dmesg            lastlog         mail.log.6      syslog
  .1
5 auth.log.0        dmesg.0          lpr.log         mail.log.6.gz   syslog
  .2.gz
6 [...]
```

Crearea jurnalelor

`sysklogd` este un utilitar care permite lucrul cu mesajele sistem și cele generate de kernel. Pe scurt, aplicațiile trimit diferite mesaje către **`sysklogd`** iar acesta, utilizând un fișier de configurare, depune aceste mesaje în anumite fișiere, le scrie în anumite console sau eventual le trimite către o altă mașină.

În Debian utilitarul **`sysklogd`** este disponibil în pachetul cu același nume.

Fișierul de configurare este `/etc/syslog.conf`. Acest fișier conține reguli care definesc modul în care sunt stocate mesajele de jurnalizare transmise de serviciile sistemului.

Deși este foarte util pentru monitorizarea sistemului, **`sysklogd`** prezintă și unele dezavantaje. De exemplu, o aplicație neglijentă ar putea genera foarte multe mesaje, ceea ce ar putea duce la umplerea discului și la instabilitatea întregului sistem. Trimirea mesajelor către o altă mașină, mai puțin importantă în rețea, ar rezolva această problemă.

Unele aplicații, precum serverul web Apache, nu folosesc **`sysklogd`**, ci implementează module proprii de jurnalizare.

Inspectarea jurnalelor

Pentru citirea jurnalelor se poate folosi orice utilitare de editare sau vizualizare, de la vi la less. Datorită faptului că mesajele recente sunt atașate la sfârșitul fișierului jurnal, un utilitar des folosit este **`tail`**.

[1] http://help.yahoo.com/l/us/yahoo/search/webcrawler/

Pentru inspectarea dinamică a jurnalelor se folosește comanda **tail** cu opțiunea `-f`. Comanda **tail** afișează în mod implicit ultimele 10 linii dintr-un fișier dat. Prin folosirea parametrului **-f** se asigură afișarea dinamică a liniilor adăugate la respectivul fișier.

Comanda de mai jos este un exemplu de folosire pentru inspectarea dinamică a activității serviciilor de e-mail:

```
root@cursuri:/var/log# tail -f /var/log/mail.log
```

Rotirea jurnalelor

Pentru a evita problema ocupării spațiului de pe disc de fișierele jurnal, sistemele Unix dispun de utilitarul **logrotate**. Acesta permite reutilizarea fișierelor de jurnalizare zilnic, săptămânal sau dacă au depășit o anumită dimensiune. În momentul în care condiția de timp sau spațiu a fost îndeplinită se efectuează o rotație.

O rotație înseamă salvarea conținutul jurnalului într-un fișier de backup cu un nume format din numele original la care se adaugă un număr. Fișierul de backup copia având un număr mai mare cu cât este mai veche (test.log.1, test.log.2 etc). La fiecare rulare, logrotate va șterge fișierul cel mai vechi și le va actualiza pe celelalte corespunzător. În general, fișierele de backup sunt arhivate și comprimate.

Mai jos sunt prezentate fișierele de jurnalizare pentru un server web Apache:

```
root@swarm:~# ls -l /var/log/apache2/
total 35092
-rw-r----- 1 root adm   8178409 Sep 19 20:45 access.log
-rw-r----- 1 root adm  10545752 Sep 13 03:35 access.log.1
-rw-r----- 1 root adm    549344 Jul 12 03:36 access.log.10.gz
-rw-r----- 1 root adm    712546 Jul  5 03:33 access.log.11.gz
-rw-r----- 1 root adm    845592 Jun 28 03:35 access.log.12.gz
[...]
```

logrotate este de obicei rulat zilnic de către serviciul de planificare periodică (`cron`). Fișierul său de configurare este `/etc/logrotate.conf`.

Alte utilitare de jurnalizare

Utilitarul **dmesg** poate fi folosit pentru afișarea informațiilor de la inițializarea sistemului. Acestea sunt mesaje ale nucleului și pot fi obținute și prin vizualizarea directă a jurnalelor de nucleu. Pentru configurațiile implicite, jurnalele de nucleu sunt păstrate în fișierul `/var/log/kern.log`.

10.3.3 Limitarea drepturilor

Sistemele Unix oferă suport în principal doar pentru două niveluri de privilegiu: utilizatori neprivilegiați și administrator (root). În practică este necesar un grad mai ridicat de flexibilitate în acordarea privilegiilor, pentru a permite utilizatorilor obișnuiți rularea unor aplicații privilegiate. În același timp, un utilizator obișnuit poate abuza de resursele care

îi sunt puse la dispoziție și poate, accidental sau intenționat, conduce la destabilizarea sau suspendarea sistemului.

De accea, un sistem de operare va oferi facilități pentru limitarea sau extinderea privilegiilor la nivel de aplicație sau la nivel de utilizator.

Folosirea atributelor setuid și setgid

O categorie aparte de aplicații o reprezintă utilitarele ce necesită resurse speciale din nucleu. Pentru a putea da posibilitatea și utilizatorilor neprivilegiați de a rula astfel de programe, identificatorul utilizatorului (uid) se schimbă pe parcursul execuției programului. La încheierea unei astfel de acțiuni se revine la identificatorul utilizatorului ce a lansat aplicația.

Într-un sistem Linux, acest lucru este realizat prin folosirea atributelor `setuid` și `setgid`. Aceste atribute sunt marcate ca permisiuni de acces pentru un fișier executabil. În momentul rulării acelui executabil, procesul creat rulează nu mai rulează cu drepturile utilizatorului care a lansat programul, ci cu drepturile utilizatorului ce deține fișierul; dacă acest utilizator este root, atunci procesul are drepturi privilegiate.

Pentru activarea atributelor `setuid` și `setgid` se folosește utilitarul **chmod**. Biții asociați celor două atribute se găsesc într-un număr în octal care precede cele 3 numere în octal asociate drepturilor de acces:

```
1  root@kiwi:~# ls -l a.out
2  -rwxr-xr-x  1 razvan razvan 13564 Jul  9 20:49 a.out
3
4  root@kiwi:~# chmod 2755 a.out
5
6  root@kiwi:~# ls -l a.out
7  -rwxr-sr-x  1 razvan razvan 13564 Jul  9 20:49 a.out
8
9  root@kiwi:~# chmod 4755 a.out
10
11 root@kiwi:~# ls -l a.out
12 -rwsr-xr-x  1 razvan razvan 13564 Jul  9 20:49 a.out
```

În exemplul de mai sus, prima rulare a comenzii **chmod** activează bitul asociat `setgid` (folosind valoarea octală 2). A doua rulare a comenzii **chmod** activează bitul asociat `setuid` (folosind valoarea octală 4). Alternativ, bitul asociat `setgid` poate fi activat folosind sintaxa `g+s`, iar bitul asociat `setuid` cu ajutorul sintaxei `u+s` ca argumente la **chmod**.

Trebuie avut în vedere că marcarea unui executabil cu dreptul de suid reprezintă un potențial risc de securitate. Dacă pe parcursul execuției utilizatorul reușește să forțeze terminarea anormală a programului înainte de refacerea uid-ului, se poate obține acces la un shell de root. Din acest motiv un pas important în securizarea unui sistem este determinarea tuturor executabilelor ce au activat atributul `setuid`, și dezactivarea acestuia de la aplicațiilor nefolosite. Aceast lucru se poate realiza cu ajutorul comenzii **find** (vezi secțiunea 12.6.3).

```
1  root@kiwi:~# find / -perm -4000 -o -perm -2000 -type f -print
```

Printre utilitarele care au nevoie de activarea atributului `setuid` pentru a putea fi rulate de către utilizatori obișnuiți se află **`ping`** și **`traceroute`**.

Schimbarea rădăcinii sistemului de fișiere (chroot)

Unul dintre mecanismele extrem de eficiente în securizarea unor servicii se numește **chroot**. Acesta presupune dimininuarea vizibilității unui serviciu în raport cu sistemul de fișiere.

Folosirea mecanismului **chroot** are rolul de reducere a riscurilor de securitate. Dacă un atacator obține acces cu ajutorul unei vulnerabilități într-un proces ce folosește **chroot**, acesta va avea acces limitat la sistemul de fișiere. Noul director rădăcină vizibil procesului poartă numele de *chroot jail*.

Pe de altă parte, folosirea acestui mecanism este destul de dificilă, datorită faptului că în general programele sunt compilate pentru a folosi biblioteci (vezi secțiunea 11.4). Din această cauză, bibliotecile trebuie incluse în directorul văzut de proces ca rădăcină. O altă soluție este compilarea statică a programelor, fără biblioteci.

Fie serverul de DNS **bind9**. Dacă se modifică scriptul de inițializare adăugând opțiunea `-t /var/dns`, serviciul va porni considerând `/var/dns/` ca rădăcină a întregului sistem de fișiere. Primul fișier căutat va fi `/etc/bind/named.conf`, astfel că înainte de lansarea serviciului va trebui copiat fișierul `/etc/bind/named.conf` în `/var/dns/etc/bind/named.conf`. Pe lângă acest fișier vor trebui copiate fișierele de zonă (necesare configurării serviciului de DNS), fișierele de jurnalizare, dar și unele fișiere speciale gen `/dev/null` sau `/dev/log`.

Majoritatea programelor ce implementează serviciile UNIX oferă posibilitatea rulării procesului cu sistemul de fișiere limitat. Dacă programul nu oferă această posibilitate, sau dacă se dorește testarea mecanismului, se poate folosi utilitarul **`chroot`**.

Utilitarul **`chroot`** este implementarea la nivelul shellului a mecanismului de schimbare a rădăcinii sistemului de fișiere văzut de un proces.

În exemplul de mai jos se rulează utilitarul chroot într-un nou director `/root/test`. Dacă nu se specifică ce comandă se rulează, implicit se execută interpretorul de comenzi precizat de variabila de mediu `SHELL`. Cum `/bin/sh` este o legătură către `/bin/bash`, este necesară copierea sa în viitorul sistem de fișiere.

```
1  root@kiwi:~# pwd
2  /root
3
4  root@kiwi:~# mkdir test
5
6  root@kiwi:~# mkdir test/bin
7
8  root@kiwi:~# cp /bin/bash test/bin/
9
10 root@kiwi:~# chroot /root/test
11 chroot: cannot run command '/bin/bash': No such file or directory
12
13 root@kiwi:~# ldd /bin/bash
14         libncurses.so.5 => /lib/libncurses.so.5 (0x4001c000)
15         libdl.so.2 => /lib/tls/libdl.so.2 (0x4005b000)
```

```
16          libc.so.6 => /lib/tls/libc.so.6 (0x4005f000)
17          /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
18  root@kiwi:~# mkdir test/lib
19
20  root@kiwi:~# cp /lib/libncurses.so.5 /lib/libdl.so.2 test/lib
21
22  root@kiwi:~# cp /lib/libc.so.6 test/lib
23
24  root@kiwi:~# chroot /root/test
25
26  root@kiwi:/# pwd
27  /
```

Se observă că, în primă fază, rularea comenzii **bash** prin **chroot** nu poate fi realizată. Motivul este absența bibliotecilor. Se afișează apoi lista bibliotecilor necesare **bash**, iar respectivele biblioteci sunt apoi copiate în viitorul sistem de fișiere. Comanda **chroot** poate fi acum executată, rezultând un prompt într-un sistem de fișiere ce are doar 2 directoare și 4 fișiere. Se pot totuși rula comenzile interne shellului **bash**.

sudo

Utilitarul **sudo** permite utilizatorilor neprivilegiați să ruleze unele comenzi având identitatea (uid-ul) altui utilizator (în general a utilizatorului root). Utilitarul **sudo** poate fi folosit pentru a oferi privilegii limitate anumitor utilizatori și a evita astfel folosirea contului root; privilegiile acestuia pot deveni o problema în cazul în care contul este compromis.

Controlul utilizatorilor care au dreptul să ruleze comenzi privilegiate, precum și lista exactă a comenzilor este păstrată în fișierul **/etc/sudoers**. Fișierul are în mod implicit doar drepturi de citire.

Pagina de manual menționează *The sudoers file should always be edited by the visudo*. Utilitarul **visudo** evită oferirea unor drepturi provizorii de scriere și minimizează astfel alte riscuri de securitate.

La lansarea unei comenzi **sudo**, după verificarea fișierului **/etc/sudoers** va trebui introdusă parola utilizatorului ce a lansat comanda **sudo**. Dacă se dorește rularea comenzilor privilegiate configurate **fără** solicitarea parolei (risc de securitate), se folosește tag-ul `NOPASSWD` în fișierul de configurare:

```
1  razvan  ALL=(ALL) NOPASSWD: ALL
```

Listingul de mai sus este o linie din fișierul `/etc/sudoers`. Utilizatorul `razvan` poate rula în numele **tuturor** utilizatorilor din sistem (`ALL=(ALL)`) orice comandă (`ALL`), fară a îi fi solicitată parola (`NOPASSWD`).

Pentru exemplul următor să presupunem că a fost dezactivat atributul `setuid` din cadrul executabilului `/bin/ping`. Pentru a permite utilizatorului `rookie` să folosească comanda **ping**, administratorul va trebui să adauge în fișierul `/etc/sudoers` următoarea linie:

```
1  rookie ALL=(ALL) /bin/ping
```

Dacă utilizatorul `rookie` încearcă rularea unui interpretor de comenzi cu drepturi privilegiate, operația eșuează:

```
rookie@kiwi:~$ sudo /bin/bash
Sorry, user rookie is not allowed to execute '/bin/bash' as root on
localhost.
```

Dacă se încearcă rularea comenzii **ping**, fară a folosi **sudo**, se obține din nou mesaj de eroare:

```
rookie@kiwi:~$ /bin/ping 141.85.99.1
-su: /bin/ping: Permission denied
```

Folosirea comenzii **sudo** impune parcurgerea fișierului `/etc/sudoers` și permite rularea executabilului `/bin/ping`:

```
rookie@kiwi:~$ sudo /bin/ping 141.85.99.1
PING 141.85.37.1 (141.85.37.1): 56 data bytes
64 bytes from 141.85.37.1: icmp_seq=0 ttl=64 time=0.5 ms
```

Rularea unei comenzi **sudo** fără a avea o corespondență în fișierul `/etc/sudoers` va fi jurnalizată ca un incident de securitate. Pentru a verifica lista comenzilor ce pot fi rulate ca `root` se poate folosi opțiunea `-l` a comenzii **sudo**:

```
rookie@kiwi:~$ sudo -l
User rookie may run the following commands on this host:
    (ALL) /bin/ping
```

Limitarea resurselor folosite de un proces sau utilizator

Utilitarul **ulimit** permite controlul resurselor alocate pentru un proces pornit din interpretorul de comenzi. Pentru inspectarea limitărilor curente se poate se folosește argumentul `-a`:

```
root@kiwi:~# ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 16382
max locked memory       (kbytes, -l) 64
[...]

root@kiwi:~# ulimit -n 2048

root@kiwi:~# ulimit -n
2048
```

În exemplu de mai sus a fost realizată o listare a tuturor limitărilor curente, pentru ca apoi să fie redefinit numărul maxim de fișiere deschise la 2048.

Dacă, în scopul prevenirii unui atac DoS local (cauzat de crearea unui număr de procese extrem de mare), se dorește limitarea numărului de procese pentru utilizatorul privilegiat, se recomandă adăugarea în fișierul `/root/.bashrc` a unei limite de procese:

```
ulimit -u 1024
```

Utilitarul **ulimit** permite limitarea drepturilor la nivelul unui proces fără a fi însă persistente. La repornirea sistemului, configurările realizate se vor pierde. Pentru a

asigura persistența schimbărilor, pe un sistem Linux se poate folosi fișierul `/etc/security/limits.conf`. Acest fișier reprezintă un mecanism eficient pentru limitarea resurselor disponibile unui utilizator sau unui grup de utilizatori.

Se pot defini două tipuri de limite, denumite limite software și limite hardware. La depășirea unei limite software se generează un mesaj de atenționare. La depășirea unei limitări hardware procesul în cauză este terminat.

Sintaxa fișierului se bazează pe definirea de reguli. O regulă are patru elemente:

- **ținta**: utilizatorii sau grupurile pentru care se aplică respectiva regulă;
- **tipul de regulă**: regulile pot fi definite ca software sau hardware;
- **tipul de limitare**: este precizat parametrul care este limitat;
- **limitarea**: valoarea la care se realizează limitarea.

Ca exemplu, se presupune că se dorește limitarea numărului de procese ale utilizatorului `rookie` la 35, iar la depășirea limitei de 20 de procese să fie generat un mesaj de avertisment. În plus se dorește limitarea numărului de autentificări **simultane** permise utilizatorului `rookie` la 10. Pentru aceasta vom adăuga în fișierul `/etc/security/limits.conf` următoarele trei linii:

```
1 rookie  soft    nproc           20
2 rookie  hard    nproc           35
3 rookie  -       maxlogins       10
```

Limitarea spațiului ocupat pe disc

Limitarea spațiului ocupat de fișierele unui utilizator se realizează prin folosirea de cote. În Linux, acest lucru este realizat cu ajutorul utilitarului **`quota`**. Întrucât configurarea acestui utilitar depășește domeniul acoperit de această carte, recomandăm cititorului să parcurgă unul din multiplele tutoriale din Internet legat de configurarea cotelor în Linux.

10.4 Atacuri de rețea

Privite ca forma principală de acțiune a crackerilor, atacurile de rețea nu trebuie să constituie cunoștințe interzise sau privite cu suspiciune. Înțelegerea modului în care un atac este produs, a condițiilor care facilitează producerea acestuia și a efectelor ulterioare sunt esențiale pentru a proteja o rețea de calculatoare sau pentru a defini politicile de securitate asociate. Cunoștințele și deprinderile legate de atacurile de rețea sunt necesare *ethical hacker*-ilor, persoane specializate în forme de testare a sistemelor informatice pentru securizarea acestora.

10.4.1 Tipuri de atacuri în rețea

Atacurile de rețea sunt clasificate în funcție de gravitate în trei categorii: atacuri de recunoaștere, atacuri DoS, și atacuri de obținere a accesului.

Atacurile de recunoaștere nu sunt atacuri propriu-zise, ci de obicei constituie faza inițială, de investigare, a unui atac (vezi secțiunea 10.4). Cel mai simplu atac de recunoaștere poate fi generat printr-un script ce inventariază stațiile dintr-o rețea folosind pachete de ping. Pentru a obține informații în legătură cu serviciile rulate pe o mașină se poate folosi o aplicație de scanat porturi (vezi secțiunea 10.4.3). Aplicațiile mai complexe de recunoaștere oferă nu numai lista serviciilor, dar și o listă a posibilelor vulnerabilități. Un astfel de utilitar este Nessus[1].

Atacurile DoS (*Denial of Service*) au ca scop congestionarea unei rețele, a unui server sau doar a unui serviciu, în scopul reducerii sau opririi funcționalității. Se poate distinge între atacurile clasice DoS, bazate în general pe *flooding*, și atacurile DoS distribuite, numite **DDoS**, ce au la bază un mecanism de propagare (un virus) pe un număr cât mai mare de destinații și sincronizarea atacului DoS către o țintă.

Un **flood** reprezintă un număr foate mare de pachete venite într-un interval scurt de timp. Un trafic ce poate fi interpretat drept flood la nivelul ruterului de acces în rețeua locală poate fi tratat drept trafic normal în nucleul Internetului. Pentru tehnologiile actuale un flood este definit astfel:

- la nivelul unui switch de nivel 2 un număr de 100.000-150.000 de pachete pe secundă;
- la nivelul unui ruter de acces (ce conectează o rețea locală) un flood are 8.000-12.000 de pachete pe secundă;
- în nucleul Internetului un trafic de milioane de pachete pe secundă poate fi tratat drept trafic legitim.

În ciuda capacității de prelucrare a unui volum de pachete de ordinul zecilor de mii pe secundă, complexitatea adăugată de modele de flood specifice poate reduce limita de pachete pe secundă la câteva sute.

ICMP flooding se bazează pe trimiterea unui număr foarte mare de pachete ICMP (de tip ping), consumând întreaga lățime de bandă disponibilă. Cel mai adesea atacul este prevenit prin filtrarea întreg traficului ICMP, cu costul pierderii funcționalități utilitarelor **`ping`** și **`traceroute`**.

Atacul **UDP flooding** are efecte puternice când folosește drept țintă porturile 7 și 19, adică serviciile de `echo` și `chargen`. Aceste servicii fiind rar folosite, în rețele locale pot fi oprite de firewall-ul de intrare în LAN, fără un impact real asupra funcționării rețelei.

Atacurile bazate pe flooding sunt în general combinate cu un atac de tip **spoofing**, prin care se generează adrese sursă fictive și diferite pentru pachetele din flood. În lipsa unui atac spoofing, o filtrare pe baza depășirii unui număr limită de pachete per sursă ar elimina atacul flooding. Atacul spoofing asigură în același timp ascunderea identității atacatorului. În plus, dacă destinația atacului bazat pe flooding nu este o sursă țintă ci adresa de difuzare a unei rețele, toate echipamentele din respectiva rețea vor încerca să răspundă unei surse ce nu există în realitate, atacul devenind un atac de tip DDoS și se va numi un **atac smurf**.

Atacurile DDoS pornesc de la puterea enormă de calcul disponibilă în rețelele actuale locale. Unele dintre aceste atacuri pot fi neintenționate, precum în cazul Slashdot effect. Numele vine de la vestitul site Slashdot, care a postat un link către un site cu capabilități

[1] http://www.nessus.org/nessus/

mai mici. Când foarte mulți utilizatori au încercat să acceseze respectivul site, efectul a fost echivalent cu un atac DoS.

Atacurile de obținere a accesului au impactul cel mai mare, deoarece prin obținerea accesului la un serviciu cel mai adesea pot fi compromise și alte servicii sau mașini.

O pondere importantă în atacurile de obținere a accesului o au atacurile bazate pe exploatarea vulnerabilității aplicațiilor web. Subiectul este unul amplu, iar cei ce vor să urmărească numeroasele metode de exploatare a riscurilor de securitate ar trebui să parcurgă cartea lui Joel Scambray, *Hacking Exposed Web Applications* [24].

Atacurile de obținere a accesului pot avea drept țintă și tehnologiile din spatele aplicațiilor web, un număr important de atacuri fiind direcționat împotriva bazelor de date. Cel mai cunoscut astfel de atac este **inserarea de cod SQL** (*SQL Injection*). Acest atac se bazează pe modul direct de interogare a bazei de date.

```
1 SELECT X from TABLE where user = $user_input AND pass = $pass_input
```

Presupunând scriptul de mai sus, dacă la introducerea în câmpul utilizator a unui nume valid de utilizator urmat `OR --` (în SQL `--` definește un comentariu), operația de selecție devine:

```
1 SELECT X from TABLE where user = $xxx OR -- AND pass = 'nu_conteaza'
```

și va fi validată pertru orice nume de utilizator aflat în baza de date. Mergând mai departe se poate introduce în câmpul utilizator `xxx OR 1=1 OR --`, aceasta garantând accesul indiferent dacă utilizatorul `xxx` există sau nu în baza de date.

Pentru a preveni un astfel de atac, interogarea bazei de date trebuie făcută prin funcții de bibliotecă și nu direct prin select.

Există aplicații ce oferă posibilitatea lansării unui număr mare de atacuri. O astfel de aplicație este metasploit[1]. După o fază de determinare a vulnerabilităților (efectuată, spre exemplu cu Nessus) se pot selecta vulnerabilitățile găsite și pot fi lansate simultan mai multe atacuri de obținere a accesului.

Dincolo de noțiunile tehnice legate de atacurile de rețea, nu trebuie pierdut din vedere riscul de securitate adus de utilizatorii neglijenți din rețea. Chiar și în cazul unor utilizatori responsabili, nu trebuie ignorate riscurile unui atac bazat pe inginerie socială. Nu este o întâmplare că cel mai cunoscut cracker este Kevin Mitnik, ale cărui atacuri s-au bazat în mare măsură pe o bună înțelegere a modului de gândire a utilizatorilor și a administratorilor din rețelele actuale.

10.4.2 Viruși, viermi, troieni

Un virus este un program sau doar o secvență de cod ce se atașează altor fișiere executabile fără cunoștința sau acceptul utilizatorului. Un virus include alături de mecanismele de execuție și modalități de replicare prin inserare în codul altei aplicații.

[1] http://www.metasploit.com

Clasificarea virușilor în funcție de modul de replicare distinge între mai multe tipuri de viruși, dintre care cei mai întâlniți sunt: viruși de email, de macro, bombe logice, viruși de boot, viermi și troieni.

Simpla prezență a unui fișier infectat pe un calculator nu este echivalentă cu infectarea sistemului. Pentru a infecta sistemul, un virus trebuie să fie executat cel puțin o dată. Unele aplicații rulează cod executabil în mod automat (fără informarea utilizatorului). Un exemplu este rularea fișierului `autorun.inf`[1] la montarea unui CD-ROM sau USB stick.

Email-ul este unul dintre modurile principale de propagare a virușilor. **Virușii de email** sunt conținuți în atașamentul emailului, deseori având extensia schimbată (cel mai adesea în extensie specifică imaginilor). Aplicația țintă a unui astfel de virus este clientul de email. Datorită numărului mare de utilizatori ai Microsoft Outlook, o mare parte a acestor viruși sunt dezvoltați special pentru această aplicație.

Odată executat un atașament de email infectat, virusul se încarcă în memoria RAM și va urmări să se multiplice. Pentru aceasta va căuta lista de adrese construite de clientul de email (*address book*). Majoritatea clienților de email creează lista de adrese în mod dinamic, fără consultarea utilizatorului, și nu o protejează prin criptare. Odată deschisă lista de adrese, virusul va folosi direct API-ul de trimis emailuri pentru a se multiplica. Exemplele de viruși de email sunt `MyDoom`, `I love you`.

O a doua categorie importantă de viruși o reprezintă **virușii de macro**. Un macro este o bucată de cod executabil, atașată unui fișier document. Aplicațiile țintă a acestor viruși sunt cele de gestionare a documentelor, gen editoare de text, foi de lucru, sau editoare de prezentări. Aceste aplicații în general vor atenționa utilizatorul înainte de deschiderea unui document ce conține macro-uri. Pentru a reduce gradul de risc, dacă la deschiderea unui fișier sunteți avertizați de existența unor macro-uri, închideți documentul și rulați un program antivirus pe respectivul fișier.

Virușii de boot nu reprezintă o amenințare majoră pentru sistemele actuale datorită dificultății de lansare în execuție. Pentru a lansa un astfel de virus, sistemul trebuie să încerce inițializarea de pe o partiție infectată. Impactul acestor viruși s-a diminuat odată cu dispariția floppy discurilor. În prezent virușii de boot se pot propaga prin CD-uri sau flash carduri.

Troienii (trojan horses) reprezintă aplicații (aparent legitime) ce conțin (sau instalează) un program malițios. Pentru a evita astfel de atacuri se recomandă instalarea aplicațiilor doar din surse sigure, precum și verificarea integrității fișierelor înainte de instalare.

Viermii (worms) sunt programe care folosesc vulnerabilități în diferite servicii din Internet pentru a se multiplica. Virușii de mail reprezintă o categorie de viermi.

Pentru a reduce impactul virușilor sunt importante atât măsurile luate la nivelul rețelei, cât și exersarea unor deprinderi din partea utilizatorilor. Este important ca accesul în rețea să fie protejat de un firewall (vezi secțiunea 10.5.1), ca serverul de email să aibă instalat un antivirus, să existe o politică de monitorizare a serviciilor etc. La nivelul utilizatorului este important să nu lanseze în execuție fișiere primite prin email, să nu deschidă atașamente primite de la necunoscuți, să mențină un antivirus actualizat pe stația de lucru etc.

[1] http://en.wikipedia.org/wiki/AutoRun#Attack_vectors

10.4.3 Scanarea porturilor

Un program de scanat porturi este o aplicație care inițiază conexiuni către toate porturile unei sistem țintă. Scopul este cel de a detremina care porturi sunt deschise, aflând astfel ce servicii sunt disponibile.

Programele de scanat porturi fac parte din setul de utilitare de bază folosite atât de crackeri (*black-hat hackers*) și ethical hackeri (*white-hat hackers*) pentru a detecta serviciile deschise pe un sistem și versiunea acestora. Dacă în urma scanării de porturi se detectează servicii potențial vulnerabile, se poate trece la folosirea unui exploit.

Sfârșitul anilor '90 în România a dus la o creștere dramatică a popularității programelor de scanat porturile, atât în rândul crackerilor cât și al administratorilor de rețea. Principala cauză a acestei schimbări a fost dezvoltarea rețelei metropolitane, precum și a diferenței foarte mari între costul traficului metropolitan și cel al traficului Internet. Astfel într-un interval relativ scurt numărul atacurilor asupra serverelor aflate în rețele publice (mai ales asupra celor aflate în universități) a crescut foarte mult.

Deși nu este ilegală, folosirea unui scaner de porturi poate fi interpretată drept un atac, și prin urmare este posibil să vă fie filtrat accesul din rețeaua din care au fost inițiate scanările. O astfel de restricționare este totuși destul de improbabilă.

Utilitarul nmap

Există numeroase aplicații de scanat porturile, una dintre cele mai cunoscute fiind **nmap**[1]. Datorită popularității de care se bucură în rândul administratorilor de sisteme Linux, aplicația a fost portată și pe platforme Windows[2].

nmap implementează mai multe metode de scanare, putând oferi informații despre porturile deschise, dar și despre porturile explicit filtrate. În plus față de informațiile referitoare la sericiile rulate, nmap pune la dispoziția administratorului de rețea metode de detectare a stațiilor active dintr-o rețea, de aflare a versiunii sistemului de operare, versiunile serviciilor active, utilizatorii ce rulează aceste servicii, precum și multe alte informații relevante pentru sistemul scanat.

Dintre cele opt tipuri importante de scanare, șase se referă la scanarea porturilor TCP, unul la scanarea porturilor UDP și unul la monitorizarea conectivității, această ultimă metodă încercând să determine doar dacă stația destinație este activă și dacă se poate ajunge la ea.

Opțiunea `-sP` (*Ping Scan*) permite detectarea stațiilor active într-o rețea locală:

```
1 root@cursuri:~# nmap -sP 141.85.37.0/24
2
3 Starting Nmap 4.62 ( http://nmap.org ) at 2009-09-23 16:21 EEST
4 Host csr.cs.pub.ro (141.85.37.1) appears to be up.
5 MAC Address: 00:09:6B:89:06:67 (IBM)
6 Host ns.catc.ro (141.85.37.2) appears to be up.
7 MAC Address: 00:17:31:49:3A:E4 (Asustek Computer)
```

[1] http://www.insecure.org/nmap

[2] http://www.sourceforge.net/projects/nmapwin

```
Host prof.cs.pub.ro (141.85.37.3) appears to be up.
MAC Address: 00:09:6B:89:05:24 (IBM)
Host info.cs.pub.ro (141.85.37.6) appears to be up.
MAC Address: 00:13:8F:78:42:2C (Asiarock Incorporation)
Host 141.85.37.7 appears to be up.
MAC Address: 00:50:56:9A:33:46 (VMWare)
Host 141.85.37.8 appears to be up.
MAC Address: 00:09:6B:89:06:67 (IBM)
```

Metodele de scanare a porturilor TCP sunt diferențiate prin prezența unui anumit flag într-un pachet TCP. Opțiunea `-sS` (*TCP SYN scan*) este opțiunea implicită care folosește flagul `SYN` al TCP și detectează porturile deschise pe un sistem.

```
razvan@valhalla:~/carte-uso$ nmap -sS localhost
You requested a scan type which requires root privileges.
QUITTING!

root@valhalla:~# nmap -sS localhost

Starting Nmap 4.68 ( http://nmap.org ) at 2009-09-23 15:38 EEST
Interesting ports on localhost (127.0.0.1):
Not shown: 1705 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
25/tcp   open  smtp
53/tcp   open  domain
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
631/tcp  open  ipp
5900/tcp open  vnc
7634/tcp open  hddtemp

Nmap done: 1 IP address (1 host up) scanned in 0.312 seconds
```

Opțiunea `-sS` necesită drepturi privilegiate. În exemplul de mai sus, utilizatorul neprivilegit `razvan` nu a putut rula comanda **`nmap`** cu opțiunea `-sS`, fiind necesară folosirea utilizatorului `root`.

Pentru utilizatorii neprivilegiați, se pot folosi alte opțiuni precum `-sT` (*TCP connect scan*). Comportamentul este similar opțiunii `-sS`, dar mai lent:

```
razvan@valhalla:~/carte-uso$ nmap -sT localhost

Starting Nmap 4.68 ( http://nmap.org ) at 2009-09-23 16:00 EEST
Interesting ports on localhost (127.0.0.1):
Not shown: 1705 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
25/tcp   open  smtp
53/tcp   open  domain
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
631/tcp  open  ipp
5900/tcp open  vnc
7634/tcp open  hddtemp
```

```
17
18 Nmap done: 1 IP address (1 host up) scanned in 0.170 seconds
```

Alte două opțiuni de scanare importante sunt `-sV` și `-O`. Opțiunea `-sV` oferă informații despre versiunea serviciilor ce ascultă pe porturile deschise, în vreme ce opțiunea `-O` oferă informații despre sistemul de operare instalat, precum și timpul de la ultima repornire.

```
1  root@kiwi:~# nmap -sV -O 141.85.37.53
2
3  Starting nmap 3.55 ( http://www.insecure.org/nmap/ ) at 2004-08-27 17:23
   EEST
4  Interesting ports on dhcp-53.cs.pub.ro (141.85.37.53):
5  (The 1659 ports scanned but not shown below are in state: closed)
6  PORT   STATE SERVICE VERSION
7  22/tcp open  ssh     OpenSSH 3.8.1p1 (protocol 1.99)
8  Device type: general purpose
9  Running: Linux 2.4.X|2.5.X|2.6.X
10 OS details: Linux 2.5.25 - 2.6.3 or Gentoo 1.2 Linux 2.4.19 rc1-rc7)
11 Uptime 0.105 days (since Fri Aug 27 14:52:12 2004)
12
13 Nmap run completed -- 1 IP address (1 host up) scanned in 11.840 seconds
```

nmap oferă și alte opțiuni. Comanda de mai jos realizează scanarea porturilor cuprinse între 1 și 100 pe stația locală:

```
1  root@valhalla:~/carte-uso# nmap -p 1-100 localhost
2
3  Starting Nmap 4.68 ( http://nmap.org ) at 2009-09-23 16:14 EEST
4  Interesting ports on localhost (127.0.0.1):
5  Not shown: 96 closed ports
6  PORT   STATE SERVICE
7  21/tcp open  ftp
8  22/tcp open  ssh
9  25/tcp open  smtp
10 53/tcp open  domain
11
12 Nmap done: 1 IP address (1 host up) scanned in 0.107 seconds
```

10.5 Securizarea rețelei

Securizarea unui sistem sau a unei rețele pot presupune investiții importante atât în echipamente, cât și în software. Cu toate acestea, orice soluție de securitate nu trebuie să piardă din vedere câteva componente de bază.

Primul pas în orice soluție de securitate o reprezintă **stabilirea unor politici clare de securitate**. O astfel de politică de securitate va urmări:

- separarea domeniilor cu nivel de securitate diferit;
- definirea clară a drepturilor fiecărui utilizator;
- definirea serviciilor ce trebuie oferite de fiecare componentă a rețelei.

Odată stabilită politica de securitate va trebui restricționat traficul nedorit/nenecesar prin **configurarea politicilor de filtrare a pachetelor** și a unui firewall (vezi secțiunea 10.5.1).

Infectarea stațiilor dintr-o rețea locală poate oferi o poartă de acces unui atacator în spatele firewall-ului, reducând gradul de securitate al rețelei. Din acest motiv un administrator responsabil nu se va ocupa doar de configurarea firewall-ului și a serverelor, ci și de securizarea stațiilor de acces.

O componentă importantă a securității o reprezintă confidențialitatea. În scopul protejării unor date sensibile se recomandă configurarea criptării traficului. Operația de autentificarea trebuie să fie, în general, o operație securizată, așa cum se întâmplă în cazul SSH.

10.5.1 Firewall-uri

Firewall-urile sunt dispozitive sau aplicații care filtrează pachetele de rețea pe baza unor politici sau reguli prestabilite.

Firewall-urile au rolul de protejare a rețelei prin filtrarea traficului nedorit/nenecesar. Un firewall poate fi configurat să filtreze traficul ICMP pentru a preveni atacuri de tip flood, cererile de inițiere de conexiune în rețeaua locală sau orice trafic care nu este menționat explicit ca fiind trafic legitim.

Firewall-urile pot fi dispozitive dedicate (**firewall-uri hardware**) sau aplicații (**firewall-uri software**). Firewall-urile hardware sunt mai rapide, dar prezintă costuri mai mari. Firewall-urile software prezintă avantajul flexibilității, dar nu sunt la fel de eficiente ca firewall-urile hardware.

Aplicația firewall pe sistemele Linux este **iptables**[1]. Exceptând rolul său de filtrare a pachetelor, iptables poate fi folosit și pentru alterarea pachetelor și pentru NAT (*Network Address Translation*).

Sistemele Windows începând cu Windows XP dispun de o aplicație firewall nativă, **Windows Firewall**, dar există numeroase aplicații de tip firewall personal, precum Zone Alarm[2].

10.5.2 Criptarea informației

O tehnică folosită încă din cele mai vechi timpuri, **criptarea** informației înseamnă folosirea unui algoritm care să transforme un mesaj cunoscut într-un mesaj criptat. Aceast algoritm presupune, în general, folosirea unei `chei de criptare`. Operația inversă se cheamă **decriptare**.

Criptarea este folosită pentru a proteja datele de acces neautorizat. Datele protejate pot fi stocate la un moment dat pe un dispozitiv de suport sau pot fi date transmise prin rețea.

În domeniul IT, există două tipuri de criptare: criptarea cu chei simetrice și criptarea cu chei asimetrice.

[1] http://www.netfilter.org/

[2] http://www.zonealarm.com/security/en-us/home.htm?lid=en-us

Criptarea cu chei simetrice presupune folosirea unei chei atât pentru criptare cât și pentru decriptare. Cheia este secretă – este cunoscută doar celor două entități care comunică. Formal, cele două operații sunt descrise astfel:

```
criptare: K(M) = C
decriptare: K(C) = M
```

`K` este cheia, `M` este mesajul inițial, iar `C` este mesajul criptat. Folosirea aceleiași chei peste mesajul criptat duce la obținerea mesajului inițial.

Exemple de algoritmi de criptare cu chei simestrice sunt DES, Twofish, Blowfish, IDEA, AES. AES (*Advanced Encryption Standard*) este algoritmul adoptat oficial de guvernul SUA. Algoritmul a fost ales în urma unui proces de standardizare de 5 ani la care au participat 15 proiecte diferite.

Criptarea cu chei asimestrice folosește o pereche de chei: o cheie publică și o cheie privată. După cum reiese și din denumire, cheia privată este secretă, iar cheia publică este accesibilă de oricine. Există două utilizări pentru o pereche cheie privată/cheie publică:

- **semnături digitale** (*digital signatures*), în care cheia privată este folosită pentru criptarea/semnarea unui document, iar cheia publică este folosită pentru a verifica faptul că documentul este autentic; oricine poate face verificarea dar numai deținătorul chei private poate semna un document;
- **criptare** (*public key encryption*), în care cheia publică este folosită pentru criptarea unui mesaj, iar cheia privată este folosită pentru decriptarea mesajului; oricine poate crea mesajul criptat pentru a fi transmis posesorului cheii private, dar numai acesta din urmă poate decripta mesajul.

Formal, procesele de semnare și verificare și cele de criptare și decriptare sunt descrise mai jos:

```
semnare: p(M) = S
verificare: P(S) = M

criptare: P(M) = C
decriptare: p(C) = M
```

În cazul semnării, se folosește cheia privată `p` pentru a "cripta" mesajul `M` în mesajul `S`. Cheia publică `P` este folosita pentru decriptarea mesajului `S` înapoi în mesajul `M`. În cazul criptării/decriptării, lucrurile se desfășoară asemănător, dar se folosește cheia publică pentru criptare și cheia privată pentru decriptare.

Criptarea cu chei asimetrice este, în general, mai sigură decât criptarea cu chei simetrice: cheia nu trebuie să fie partajată și algoritmii sunt mai robuști. Cu toate acestea, criptarea cu chei asimestrice este mult mai lentă decât criptarea cu chei simetrice. Există situații în care cele două metode se folosesc la un loc. Spre exemplu, în cazul SSH, inițierea conexiunii se realizează folosind criptare cu chei asimetrice; în urma inițierea se realizează o negociere după care se obține o cheie simetrică; această cheie va fi folosită în continuare pentru criptarea comunicației între clientul și serverul SSH.

O extensie a semnăturilor digitale o reprezintă **certificatele digitale**[1]. Un certificat digital conține o cheie publică și un set de informații despre posesorul certificatului semnate digitale. Întrucât cheia publică este inclusă în certificat, oricine poate verifica validitatea informațiilor. Depinzând de infrastructura folosită, semnatura poate fi a unei autorități specializate (CA – *certificate authority*[2], în cazul PKI – *public key infrastructure*[3]) sau a mai multor utilizatori (endorsements), în cazul *web of trust*[4].

10.5.3 Monitorizarea rețelei

Autorii de cărți de rețele de calculatoare sau de ghiduri pentru administratorii de rețea precizează în general că activitatea de monitorizare este una dintre cele mai importante sarcini ale unui administrator. Cu toate acestea, adesea, recomandările propuse sunt deseori limitate la cele evidente precum: "Un administrator trebuie să citească regulat jurnalele de pe fiecare server" sau "Este important să nu uitați să porniți serviciul de monitorizare".

Este important de înțeles că monitorizarea nu se rezumă doar la rularea unui daemon de genul **sysklogd** sau la deschiderea săptămânală a unei aplicații precum **Event Viewer**. Cele două funcții principale ale monitorizării sunt asigurarea dinamică a securității prin alertarea în fața unei încercări de atac, și posibilitatea localizării și definirii unei defecțiuni sau a unei funcționări suboptimale.

Există mai multe criterii pentru a clasifica diversele tipuri de acțiuni de monitorizare. O primă clasificare face distincția între:

- monitorizarea unui sistem specific;
- monitorizarea conectivității;
- monitorizarea traficului din rețea.

Monitorizarea unui dispozitiv specific presupune rularea unui serviciu ce va urmări apelarea celorlalte servicii, precum și accesul la jurnalele create de acest server. Un exemplu este serviciul de jurnalizare oferit de **sysklogd**. **Monit**[5] permite monitorizarea proceselor, fișierelor și poate să execute diverse acțiuni la nevoie.

Monitorizarea conectivității se bazează deseori pe mecanisme foarte simple, cel mai important dintre acestea fiind folosirea utilitarului `ping`, metoda fiind denumită **ping sweep**. Crearea unui sistem de interogări ICMP periodice poate oferi administratorului informații despre întreruperea unui segment de rețea sau despre defectarea sau stingerea unuia dintre servere. Un astfel de sistem de interogări poate fi realizat ori printr-un script simplu, ori se poate apela la aplicații dedicate.

Monitorizarea de rețea cuprinde numeroase componente, de la dispozitive hardware dedicate (denumite generic **IDS** – *Intrusion Detection System*), la programe de interceptare a traficului în rețea (programe de sniffing), precum și programe de evaluare

[1] http://en.wikipedia.org/wiki/Digital_certificate
[2] http://en.wikipedia.org/wiki/Certificate_authority
[3] http://en.wikipedia.org/wiki/Public_key_infrastructure
[4] http://en.wikipedia.org/wiki/Web_of_trust
[5] http://mmonit.com/monit/

a securității unui sistem din rețea, în această categorie intrând programele de scanat porturi. O aplicație foarte cunoscută de monitorizare a rețelei este **Nagios**[1].

Apariția switchurilor în rețelele locale a transformat mediul partajat al Ethernetului într-un mediu dedicat. Cu alte cuvinte, într-o rețea cu switchuri, pachetele schimbate între două stații nu vor mai fi vizibile de o a treia stație aflată în aceeași rețea locală. În această situație, un administrator de rețea ce dorește monitorizarea traficului către o anumită destinație va trebui ca înainte de a lansa programul de interceptare a traficului să facă un atac **ARP poisoning**[2] pentru respectiva destinație. Pentru aceasta se pot folosi utilitare precum Cain[3] și dsniff[4].

Utilitarul tcpdump

tcpdump este un utilitar destinat monitorizării traficului în rețea și achiziționării de date.

Plăcile de rețea, cum sunt cele Ethernet, capturează la nivelul legătură de date doar cadrele adresate lor sau pe cele de difuzare. De aceea, pentru a captura toate cadrele, `tcpdump` trebuie să treacă interfața într-un mod special de lucru, numit *promiscuous mode*, pentru care are nevoie de suportul sistemului de operare.

La o rulare simplă, `tcpdump` capturează pachetele de pe prima interfață de rețea a sistemului și afișează informații din antetul IP și TCP.

```
1 root@frodo:~# tcpdump
2 tcpdump: listening on eth0
3 14:45:09.252803 frodo.noi.39993 > 64.12.30.90.5190: P
  444608787:444608793(6) ack 1453900500 win 32893 (DF)
4 14:45:09.254097 frodo.noi.33335 > main.noi.domain: 16713+ PTR?
  90.30.12.64.in-addr.arpa.(42) (DF)
5 14:45:09.254583 main.noi.domain > frodo.noi.33335:  16713 NXDomain 0/1/0
  (113)(DF)
```

În exemplul de mai sus, fiecare linie conține:

- timpul în care pachetul TCP a intrat în rețeaua locală;
- informații legate de antetul IP (adresa sursă și adresa destinație);
- informații din antetul TCP (port sursă, pot destinație, numere de secvență, flag-uri TCP).

Pentru a vedea și conținutul pachetului se poate utiliza opțiunea `-x`:

```
1 root@frodo:~# tcpdump -c1 -x
2 tcpdump: listening on eth0
3 18:38:51.087244 64.12.24.48.5190 > frodo.noi.32777: P
  3127656695:3127656809(114) ack 3009470569 win 16384 (DF) [tos 0x80]
4                          4580 009a c7a1 4000 6406 354d 400c 1830
5                          c0a8 000b 1446 8009 ba6c 40f7 b360 e069
6                          5018 4000 1f0d 0000 2a02 20dc 006c 0003
7                          000b 0000 87d9 0d61 0836 3032 3033 3631
8                          3400 0000 0700 0100 0200 7000 0c00 2500
```

[1] http://www.nagios.org/
[2] http://en.wikipedia.org/wiki/ARP_spoofing
[3] http://www.oxid.it/cain.html
[4] http://www.monkey.org/~dugsong/dsniff/

```
9                          0000
10 6 packets received by filter
11 0 packets dropped by kernel
```

Conform structurii unui pachet IP, primii 20 octeți (4580 009a c7a1 4000 6406 354d 400c 1830 c0a8 000b) reprezintă antetul IP iar următorii 20 de octeți (1446 8009 ba6c 40f7 b360 e069 5018 4000 1f0d 0000) reprezintă antetul TCP.

Există soluții grafice ce realizează interpretarea conținutului pachetului. Cea mai cunoscută aplicație grafică de interceptare și interpretare a traficului de rețea este **Wireshark**[1], utilitar disponibil atât în mediul Windows, cât și în Linux. Echivalentul pentru `tcpdump` în lumea Windows este `windump`[2].

10.6 Studiu de caz

10.6.1 Drepturile pe fișiere în NTFS

Sistemul de fișiere NTFS oferă o serie de îmbunătățiri față de FAT, atât din punctul de vedere al performanței, cât și al securității. Într-un sistem de fișiere FAT, toate conturile de utilizator au drepturi egale și depline asupra sistemului de fișiere.

Figura 10.2: Precizarea drepturilor în NTFS

Securitatea în NTFS se implementează prin intermediul mecanismului de liste de acces. O listă de acces reprezintă un set descriptori atașati fișierelor și directoarelor din sistemul

[1] http://www.wireshark.org/
[2] http://www.winpcap.org/windump/install/

de fișiere. Pentru ca un proces să poată accesa un fișier sau un director, sistemul de securitate verifică dacă procesul are dreptul de a-l accesa.

Ca măsură sporită de securitate, sistemul de fișiere NTFS oferă posibilitatea criptării datelor pe care le stochează. Avantajul implementării mecanismelor de criptare direct în sistemul de fișiere îl reprezintă viabilitatea pe care acest tip de securitate o menține în lucrul cu orice sistem de operare care oferă suport pentru sistemul respectiv de fișiere.

Din mediul Windows, criptarea unui fișier sau a unui director (inclusiv a conținutului său) se face din ecranul de proprietăți al fișierului/directorului (clic dreapta > Properties > Advanced > "Encrypt contents to secure data").

Pentru a putea acționa granular asupra securității sistemului de fișiere, în Windows trebuie dezactivată opțiunea "Use simple file sharing" (de la Explorer > Folder Options > View > Use Simple File Sharing: debifare). Ulterior, atribuirea drepturilor se face din fereastra de proprietăți a unui director sau a unui fișier, sub tabul `Security`. Se selectează utilizatorul sau grupul care va fi editat, iar din lista de drepturi se indică permisiunile ce vor fi atribuite. Toate configurările se aplică doar fișierului sau directorului selectat. Pentru a configura explicit permisiunile, butonul `Advanced` afișează o nouă interfață în care pot fi vizualizate toate tipurile de permisiuni ce sunt aplicate fișierului sau directorului ales, inclusiv drepturi moștenite prin ierarhia sistemului de fișiere.

10.6.2 Recuperarea parolei

Un bun administrator de sistem trebuie să fie conștient de mecanismele de recuperare a parolei pentru a putea preveni proceduri neautorizate de modificare a acesteia.

Există două metode importante pentru a realiza recuperarea parolei de root, ambele necesitând accesul fizic la echipament. Prima metodă presupune modificarea din meniul bootloader-ului a parametrilor trimiși la încărcarea nucleului, iar a doua se bazează pe folosirea unui CD Linux bootabil.

Editarea GRUB

Prima metodă de a parolei de root necesită editrea meniului de boot a GRUB. Din meniul GRUB se selectează imaginea de nucleu ce se dorește încărcată și se activează editarea respectivei linii de grub prin opțiunea `e`. În capitolul 6 sunt prezentate pe larg metodele de modificare a opțiunilor transmise la încărcarea nucleului.

La linia conținând imaginea de nucleu selectată se adaugă opțiunea `init=/bin/bash` și se continuă procesul de inițializare a sistemului. În urma încheierii procesului de inițializare se va obține un shell root. Dacă la execuția comenzii **`passwd`** se primește mesaj de eroare trebuie remontat sistemul de fișiere pentru scriere și citire. Acest lucru se realizează folosind comanda:

```
1 root@ubuntu:~# mount -o remount,rw /
```

Pentru a preveni o recuperare neautorizată a parolei de root se recomandă folosirea parolelor de acces la meniul bootloader-ului. În cazul GRUB, acest lucru se realizează cu ajutorul opțiunii `password` în cadrul fișierului de configurare a GRUB.

Folosirea unui Live CD

A doua metodă de recuperare a parolei presupune folosirea unui Live CD Linux. Dacă ordinea de boot nu acordă prioritate unității optice în fața hard discului, va trebui schimbată ordinea de boot din BIOS.

După inițializarea sistemului de pe CD trebuie obținut un shell de root. În cazul Live CD-ului Ubuntu aceasta se realizează prin executarea din contul neprivilegiat a comenzi **sudo**:

```
1 root@ubuntu:~# sudo bash
```

Trebuie creat un nou director în care se va face montarea vechiului sistem de fișiere:

```
1 root@ubuntu:~# mkdir /mnt/x
2
3 root@ubuntu:~# mount /dev/hda2 /mnt/x
```

În acest moment putem invoca comanda chroot:

```
1 root@ubuntu:~# chroot /mnt/x
```

Ultimul pas presupune invocarea utilitarului **passwd** și schimbarea parolei de root. Întrucât, prin intermediul Live CD-ului, utilizatorul are acces privilegiat la întreg sistemul, va avea acces complet la sistemul de fișiere de pe discul sistemului. Invocarea comenzii **passwd** va conduce la modificarea informației din fișierul `/etc/shadow`.

Pentru a preveni o astfel de procedură este necesar să se forțeze din BIOS ca prim dispozitiv de boot unul dintre hard discuri și să se definească o parolă de acces la meniul de configurare a BIOS-ului.

Cuvinte cheie

- cel mai mic privilegiu
- cea mai slabă verigă
- controlul accesului
- autentificare
- autorizare
- criptare
- atacuri
- chmod
- chown
- chroot
- df
- DoS
- drepturi pe fișiere în NTFS
- drepturi pe fișiere în UNIX
- du
- /etc/sudoers
- /etc/passwd, /etc/shadow
- /etc/security/limits.conf
- liste de control a accesului – ACL
- matrice de drepturi
- firewall
- criptare, decriptare
- chei simetrice, chei asimetrice
- chei private, chei publice
- semnătură digitală, certificat digital
- hacker

- netstat
- monitorizare
- nmap
- parole
- politică de securitate
- quota
- securitatea fizică
- sniffer
- sudo
- suid
- syslog
- tcpdump
- top
- ulimit
- umask
- viruși

Întrebări

1. Care dintre următoarele NU este o funcționalitate oferită de utilitarul **`nmap`**?
   - ❑ scanarea porturilor UDP și TCP
   - ❑ detectarea versiunii sistemului de operare
   - ❑ detectarea stațiilor active din rețeaua locală
   - ❑ filtrarea pachetelor de rețea
2. Fie următoarele comenzi:

```
1 umask 022
2 echo "Acesta nu este un test" > test
3 chmod go+x test
```

   Care vor fi drepturile pentru utilizatorii grupului fișierului `test`?
   - ❑ doar citire
   - ❑ doar execuție
   - ❑ citire și execuție
   - ❑ citire, scriere și execuție
3. Care dintre parolele de mai jos este cea mai sigură pentru utilizatorul `cornel`?
   - ❑ cornelush
   - ❑ coRnel
   - ❑ _cOrN31[]sh+
   - ❑ niciuna, toate parolele vor fi încercate într-un atac bazat pe dicționar
4. Care dintre riscurile de mai jos reprezintă riscuri de securitate fizică pentru un server?
   - ❑ lipsa unui firewall
   - ❑ parole pentru utilizatori prea simple
   - ❑ vârfuri de tensiune în rețeaua electrică

- ❑ absența licenței pentru sistemul de operare

5. În urma procesului de scanare de porturi se poate determina:
   - ❑ dacă un serviciu rulează pe un port dat
   - ❑ versiunea serviciului
   - ❑ dacă serviciul este filtrat
   - ❑ toate variantele
6. Primirea unui email ce conține un virus pe un sistem ce nu are antivirus duce automat la infectarea sa. Un sistem infectat cu un virus nu mai poate fi țința unui alt atac.
   - ❑ adevărat, adevărat
   - ❑ adevărat, fals
   - ❑ fals, adevărat
   - ❑ fals, fals
7. Traficul către un server este monitorizat cu **`tcpdump`**. Care dintre următoarele informații NU poate obținută?
   - ❑ portul pe care rulează serviciul
   - ❑ numărul de clienți ai server-ului
   - ❑ dimensiunea pachetelor folosite
   - ❑ adresa IP a serverului
8. Pentru a limita numărul de procese per utilizator se poate folosi:
   - ❑ **`ulimit`**
   - ❑ **`quota`**
   - ❑ **`nmap`**
   - ❑ **`chroot`**
9. Care este primul pas în securizarea unei rețele?
   - ❑ definirea unei politici de securitate
   - ❑ definirea regulilor de filtrare pe firewall
   - ❑ instalarea de programe antivirus pe toate stațiile din rețea
   - ❑ criptarea traficului
10. Pe un sistem Linux, ce fișier stochează parolele utilizatorului (criptate)?
    - ❑ `/etc/passwd`
    - ❑ `/etc/shadow`
    - ❑ `/etc/security/limits.conf`
    - ❑ `/var/log/apache2/access.log`

Capitolul 11

Compilare și linking

He who hasn't hacked assembly language as a youth
has no heart. He who does as an adult has no brain.
John Moore

Ce se învață din acest capitol?

- Cum se obține un program executabil dintr-un program sursă
- Compilarea și interpretarea unui program
- Etapele compilării
- Utilizarea GCC pentru compilarea și link-editarea unui program
- Optimizarea compilării
- Fișiere obiect și fișiere executabile
- Biblioteci de funcții
- Automatizarea acțiunilor cunoscând dependențele între ele; utilitarul make
- Concepte de portabilitate
- Link-editarea codului C și C++

11.1 Introducere

Unul dintre aspectele importante în utilizarea unui sistem de operare este dezvoltarea de noi programe. Acest lucru presupune folosirea unui limbaj de programare pentru scrierea de cod sursă. Codul sursă este compilat, iar programul executabil obținut este rulat. Acțiunea de scriere de cod sursă poartă numele de programare sau dezvoltare. Persoana care scrie codul se numește programator sau dezvoltator. Termenii în limba engleză sunt **programmer**, **software developer** sau **software engineer**.

Sistemele de operare moderne pun la dispoziția dezvoltatorului/programatorului o suită de aplicații folosite pentru scrierea de noi programe. În cazul familiei de sisteme de operare Windows, suita de aplicații Visual Studio reprezintă forma cea mai cunoscută

pentru dezvoltarea de noi aplicații. Aceasta permite crearea de programe în diverse limbaje de programare și cu utilitate diversă. Dezvoltarea de aplicații se desfășoară într-un mediu integrat (*IDE* – Integrated Development Environment) (vezi secțiunea 14.9).

De partea cealaltă, sistemele Linux pun la dispoziția utilizatorului suita de aplicații GNU, folosită pentru compilarea, link-editarea și depanarea programelor. În mod tradițional, dezvoltatorii de aplicații pe sisteme Linux folosesc utilitare dedicate (editor, compilator, depanator), dar se folosesc din ce în ce mai mult și mediile integrate precum Eclipse sau Anjuta (vezi secțiunea 14.9.1 și secțiunea 14.9.2).

11.1.1 Editoare

Prima fază a procesului de dezvoltare este scrierea de cod specific unui limbaj de programare. Codul reprezintă o înșiruire de instrucțiuni, variabile și funcții pentru acel limbaj. Limbajul impune o sintaxă pe care programatorul trebuie să o respecte. Fișierul în care se scrie codul este, în general, un fișier text și poartă numele de fișier sursă, sau mai simplu sursă. Expresia "trebuie să mă uit prin surse"[1] se referă la parcurgerea fișierelor sursă ale unei aplicații pentru analiza codului.

Scrierea de cod sursă se realizează prin intermediul unui editor. Editoare răspândite în lumea Unix sunt vi, Emacs, nano, joe, kate, gedit. Editoare răspândite în lumea Windows sunt Notepad++, Ultraedit, Crimson Editor. De obicei, în lumea Windows, editoarele sunt parte a unui IDE (Integrated Development Environment) cum este Visual Studio sau Eclipse.

Un editor este folosit pentru scrierea unui fișier sursă. Drept urmare, editoarele moderne prezintă o serie de caracteristici utile acestui scop: indentare automata, colorare a codului (syntax highlighting), code folding, variable completion, folosirea de registre de editare pentru operatii multiple etc.

Lumea Unix este dominată de rivalitatea dintre editoarele vi și Emacs[2]. În vreme ce vi este un adept al filozifiei Unix (keep it simple) și este folosit doar pentru editarea de fișiere sursă, Emacs este un editor complex cu facilități care includ posibilitatea de folosire ca mediu integrat de dezvoltare (IDE), client de e-mail, client de IRC etc. De-a lungul vremii, însă, numeroase facilități au fost adăugate și editorului vi astfel încât funcționalitățile oferite sunt apropiate. vi și Emacs reprezintă de fapt două familii de editoare. Cei mai cunoscuți reprezentanți ai celor două familii sunt, respectiv, editorul Vim[3] și GNU Emacs[4]. Ambele editoare dispun de facilități avansate pentru editare eficientă și permit posibilitatea de extensie și personalizare. Cunoscuți pentru fanatismul folosirii unuia dintre cele două editoare, programatorii vor prezenta fișierul propriu de configurare (.vimrc în cazul Vim și .emacs în cazul GNU Emacs).

O prezentare mai amplă a editorului Vim găsiți în capitolul 14.

---

[1] http://encyclopedia2.thefreedictionary.com/Use+the+Source+Luke

[2] http://en.wikipedia.org/wiki/Editor_war

[3] http://vim.sourceforge.net/

[4] http://www.gnu.org/software/emacs/

11.1.2 Compilare și interpretare

Programul scris în cod sursă este compilat sau interpretat pentru a îndeplini sarcina pentru care a fost scris. Atât compilarea cât și interpretarea unui program sunt forme de translatare a acestuia și execuție pe un sistem hardware. Translatarea se referă la transformarea într-un limbaj a unui program scris într-un alt limbaj.

În definiția cea mai simplă, compilarea unui program se referă la obținerea unui fișier binar executabil. Executarea acestui fișier duce la îndeplinirea sarcinii pentru care a fost scris programul. După cum se va observa în secțiunea 11.3 există mai multe faze specifice compilării unui program.

De partea cealaltă, interpretarea unui program elimină faza obținerii unui executabil și execută codul asociat unui program pe măsura parcurgerii acestuia. Avantajul interpretării unui program este timpul mai scurt de dezvoltare (se elimină faza de compilare) și depanarea mult mai facilă. Pe de altă parte, dezavantajul interpretării unui program este viteza mai scăzută de execuție. În cazul compilării, rularea executabilului înseamnă rularea de cod specific sistemului fizic. În cazul interpretării, codul sursă este parcurs, analizat și apoi executat, rezultând într-un timp de rulare mai mare.

Limbaje compilate și limbaje interpretate

Utilitarul folosit pentru compilarea unui program poartă numele de compilator, iar cel folosit pentru interpretare poartă numele de interpretor. Compilatorul, respectiv interpretorul, sunt **implementări** ale limbajului de programare asociat. Astfel, compilatorul/interepretorul cunoaște sintaxa limbajului de programare în care a fost scris respectivul program și este folosit pentru translatarea acestuia într-o formă executabilă: fișier executabil în cazul compilatorului, sau execuție în timpul translatării în cazul interpretorului.

Una dintre cele confuziile din lumea limbajelor de programare și a programării în general este clasificarea în limbaje compilate și limbaje interpretate. De multe ori se menționează că C, C++, Ada, Fortran, Java sunt limbaje compilate, în vreme ce Perl, Python, PHP, Ruby, Lisp sunt limbaje interpretate.

De fapt, orice limbaj poate fi atât interpretat cât și compilat. Astfel, deși limbajul C este de obicei compilat, există și interpretoare pentru acesta[1].

În concluzie, nu există limbaje interpretate sau compilate ci **implementări de compilator** sau **implementări de interpretor** pentru un limbaj. Într-adevăr, limbajele au asociate preponderent un anumit tip de implementare (C are implementări de compilator, Perl are implementări de interpretor) și, drept urmare, sunt denumite "limbaje compilate" sau "limbaje interpretate". Totuși, această clasificare este forțată și recomandăm să fie atribuită **implementării** acelui limbaj.

[1] http://www.softintegration.com/

Limbaje de nivel scăzut și limbaje de nivel înalt

Dintre multiplele clasificări posibile ale limbajelor de programare[1], o clasificare răspândită este în limbaje de programare de nivel scăzut și limbaje de programare de nivel înalt.

> **Limbajele de programare de nivel scăzut** sunt acele limbaje care nu oferă o abstractizare a arhitecturii sistemului de calcul. În mod obișnuit, aceste limbaje oferă o translatare minimă a instrucțiunilor arhitecturii.

Limbaje de programare de nivel scăzut sunt codul mașină, adică scrierea unui program direct în binar și limbajul de asamblare. Un programator nu va scrie niciodată cod mașină și foarte rar cod în limbaj de asamblare.

> **Limbajele de programare de nivel înalt** sunt limbaje de programare care ofera o abstractizare a arhitecturii sistemului de calcul. Limbajele de programare de nivel înalt sunt mai ușor portabile și oferă o sintaxă mai apropiată de limbajul natural.

Limbajele de programare de nivel înalt oferă o plajă largă de abstractizări. Unele limbaje pot ascunde complet detalii legate de modul de adresare a procesorului, alocarea/dezalocarea memoriei, lucrul cu obiecte etc. Noțiunea de limbaj de nivel înalt devine relativă raportat la facilitățile oferite. Dacă în anii '70, limbajul de programare C era considerat limbaj de nivel înalt, în zilele noastre poate fi considerat limbaj de nivel scăzut în comparație cu limbaje precum Java, Python, Ruby.

Exemple de limbaje de nivel înalt sunt: C/C++, Java, Perl, Python, PHP, Ruby, Lisp, Scheme, Prolog, Haskell.

11.1.3 De la sursă la executabil, de la executabil la proces

După cum s-a precizat, în urma compilării unui fișier sursă se obține un executabil care poate fi apoi rulat pentru îndeplinirea sarcinii pentru care a fost scris. În capitolul 5 s-a menționat că un executabil este o entitate pasivă care descrie modul în care va rula programul. Rularea efectivă se realizează în urma execuției programului în cadrul unui proces. Procesul este entitatea activă în cadrul căreia se vor aloca resursele programului și se va rula codul asociat pe procesor.

Așa cum reiese și din figura 11.1, obținerea unui executabil dintr-un fișier sursă se realizează în urma compilării acestuia din urmă. La fel, procesul este obținut din executabil în urma execuției. Utilizatorul va trebui să cunoască doar limbajul de nivel înalt și să scrie codul sursă în acest limbaj. În continuare, compilatorul va obține executabilul; executabilul este un fișier binar care descrie, în cod mașină, operațiile precizate de utilizator. În faza de execuție, cu suportul nucleului sistemului de operare, din executabilul obținut anterior se creează un proces. Procesul este entitatea activă care va executa pe procesor operațiile descrise în cod mașină în zona de cod a executabilului (vezi secțiunea 5.1.2).

[1] http://en.wikipedia.org/wiki/Categorical_list_of_programming_languages

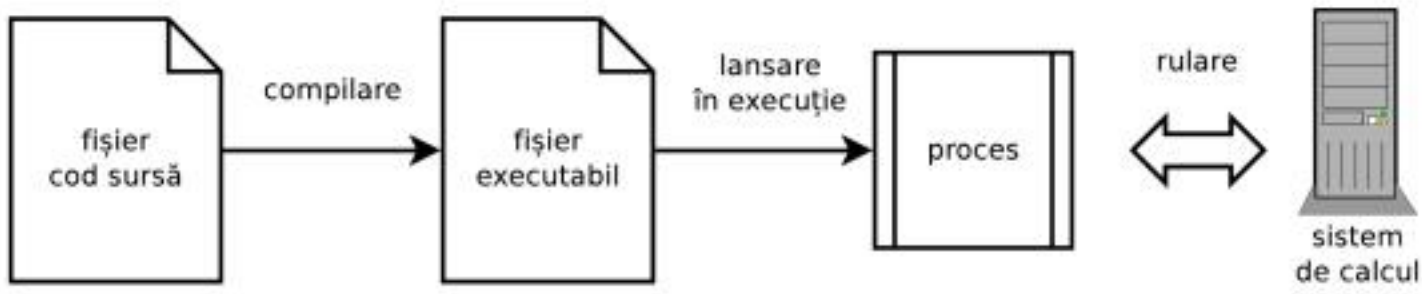


Figura 11.1: De la fișier sursă la executabil; de la executabil la proces

Compilatorul oferă în acest fel un beneficiu important: utilizatorul nu trebuie să dețină informații despre sistemului fizic din spate (zone de memorie, instrucțiuni ale procesorului etc.), ci numai noțiuni despre limbajul folosit pentru a scrie codul sursă. Compilatorul va fi responsabil cu translatarea codului sursă (fișierul sursă, text) în cod mașină (fișierul executabil, binar). Ulterior, în urma rulării executabilului, sistemul de operare creează procesul care execută codul pe procesor.

Folosirea unui compilator aduce după sine avantajul portabilității codului: utilizatorul va scrie cod sursă portabil specific limbajului de programare. Compilatorul generează cod binar neportabil specific sistemului fizic pe care acesta va rula. Mai multe detalii despre portabilitate sunt prezentate în secțiunea 11.6.

În exemplul de mai jos este prezentat un fișier sursă C (`hw.c`) și comenzile pentru compilarea acestuia (**`gcc hw.c`**), generarea unui executabil (`a.out`) și executarea acestuia în cadrul unui proces (**`./a.out`**).

```c
#include <stdio.h>

int main(void)
{
        printf("Hello, World!\n");

        return 0;
}
```

Listing 11.1: hw.c

```
mircea@cougar:~/carte-uso/cap-10$ file hw.c
hw.c: ASCII C program text

mircea@cougar:~/carte-uso/cap-10$ gcc hw.c

mircea@cougar:~/carte-uso/cap-10$ file a.out
a.out: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/
Linux 2.4.1, dynamically linked (uses shared libs), for GNU/Linux 2.4.1,
not stripped

mircea@cougar:~/carte-uso/cap-10$ ./a.out
Hello, World!
```

11.1.4 Pachete necesare

Pentru dezvoltarea de aplicații este nevoie de

- cunoștințe despre un anumit limbaj de programare;

- un editor folosit pentru a scrie codul sursă;
- un interpretor sau un compilator pentru acel limbaj.

În restul acestui capitol, limbajul de programare folosit pentru exemplificare va fi C. Compilatorul folosit va fi GCC[1], compilatorul implicit pe distribuțiile Linux. Pachetele necesare pentru dezvoltarea de aplicații C sub Ubuntu sunt `gcc`, `libc6-dev`. Se recomandă și paginile de manual pentru dezvoltare din pachetul `manpages-dev` și documentația bibliotecii standard C din pachetul `glibc-doc` (sau `glibc-doc-reference` pe sistemele Debian). Instalarea utilitarului `Make` (vezi secțiunea 11.5) este, de asemenea, recomandată.

```
1 root@cougar:~# apt-get install gcc libc6-dev manpages-dev glibc-doc make
```

Mai ușor, se poate instala pachetul virtual `build-essential`. Destinat formal creării de pachete .deb, instalarea acestui pachet conduce la instalarea pachetelor importante pentru dezvoltarea programelor pe un sistem Ubuntu. Pachetul nu include pachetele de documentație, deci acestea vor trebui instalate separat:

```
1 root@cougar:~# apt-get install build-essential manpages-dev glibc-doc
```

11.2 Compilare. GCC

În general, procesul de compilare se referă la obținerea unui fișier executabil dintr-un fișier cod sursă. Aplicația care realizează această translatare se numește compilator. Un compilator este o aplicație complexă care, pe lângă rolul de translator, trebuie să îndeplinească și alte cerințe. Codul binar obținut de compilator în cadrul executabilului va trebui să satisfacă una (sau mai multe) din următoarele solicitări:

- să fie cât mai mic;
- să ruleze cât mai rapid;
- să consume cât mai puțină memorie în momentul rulării.

Un compilator trebuie să implementeze cât mai mult din specificațiile standardelor în rigoare. Spre exemplu, în cazul C, standardele sunt stabilite de ISO[2]. Ultimul standard este ISO 9899:1999 denumit și C99[3].

Pe sistemele Unix, compilarea se realizează prin transmiterea fișierului sursă ca argument compilatorului. Compilarea rezultă în generarea executabilului implicit `a.out`[4]:

```
1 mircea@cougar:~/carte-uso/cap-10$ gcc hw.c
2
3 mircea@cougar:~/carte-uso/cap-10$ ls
4 a.out hw.c
5
6 mircea@cougar:~/carte-uso/cap-10$ ./a.out
7 Hello, World!
```

[1] http://gcc.gnu.org/
[2] http://www.iso.org/iso/home.htm
[3] http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1256.pdf
[4] http://en.wikipedia.org/wiki/A.out

Pe sistemele Windows compilarea se realizează, în linia de comandă, folosind utilitarul **cl**[1]. Executabilul obținut are același nume cu fișierul sursă însă cu extensia `.exe`:

```
C:\> cl hw.c

C:\> hw.exe
Hello, World!
```

GCC este compilatorul implicit pe distribuțiile Linux și pe un număr mare de alte sisteme Unix. Scris de Richard Stallman[2] în 1989 pentru proiectul GNU, era folosit inițial doar pentru a compila programe C (GNU C Compiler). De-a lungul timpului s-a extins, și în ziua de azi, poate compila programe scrise în C++, Fortran, Objective-C, Pascal, Java etc. Drept urmare, denumirea sa a fost schimbată în GNU Compiler Collection. Pachetul GCC oferă executabile (denumite și front-end-uri) pentru fiecare tip de limbaj pe care îl poate compila:

- **gcc** este folosit pentru a compila programe scrise în C;
- **g++** compilează programe scrise în C++;
- **gnat** compilează programe scrise în Ada;
- **gcj** compilează programe scrise în Java;
- **g77** compilează programe scrise în Fortran;

11.2.1 Utilizare GCC

La o rulare implicită a GCC, acesta generează executabilul `a.out`. Fie următorul program C:

```
#include <stdio.h>

main()
{
        printf("Hello, World!\n");
}
```

Listing 11.2: hw2.c

În forma sa cea mai simplă, compilarea se realizează prin transmiterea fișierului sursă ca argument comenzii **gcc**:

```
mircea@cougar:~/carte-uso/cap-10$ gcc hw2.c
```

În mod implicit se obține executabilul `a.out`. Pentru rularea acestuia se folosește construcția `./` care precizează execuția din directorul curent:

```
mircea@cougar:~/carte-uso/cap-10$ ./a.out
Hello, World!
```

Se poate specifica numele executabilului care se dorește obținut prin folosirea opțiunii `-o` urmată de numele executabilului:

[1] http://msdn.microsoft.com/en-us/library/ms235639(VS.80).aspx
[2] http://en.wikipedia.org/wiki/Richard_Stallman

```
mircea@cougar:~/carte-uso/cap-10$ gcc hw2.c -o my_exec

mircea@cougar:~/carte-uso/cap-10$ ./my_exec
Hello, World!
```

În general, executabilele în Unix nu au extensie. Opțiunea `-o` poate fi prezentă oriunde în linia de comandă, doar că va trebui să fie urmată de numele executabilului:

```
mircea@cougar:~/carte-uso/cap-10$ gcc -o my_other_exec hw2.c

mircea@cougar:~/carte-uso/cap-10$ ./my_other_exec
Hello, World!
```

O greșeală frecventă este folosirea opțiunii `-o` urmată imediat de numele fișierului sursă:

```
mircea@cougar:~/carte-uso/cap-10$ gcc -o hw2.c my_other_exec
my_other_exec:(.rodata+0x0): multiple definition of `_fp_hw'
/usr/lib/gcc/i486-linux-gnu/4.1.2/../../../../lib/crt1.o:(.rodata+0x0):
first defined here
[...]
```

ATENȚIE: Această greșeală are consecințe grave pentru că duce la pierderea fișierului sursă.

Programul `hw2.c` prezentat mai sus compilează, dar nu respectă întru totul standardul C. De exemplu, funcția `main` trebuie să întoarcă `int`[1]. Compilatorul nu va afișa, în mod implicit, niciun fel de avertisment pentru astfel de inconsistențe. Acest lucru poate fi schimbat prin folosirea opțiunii `-Wall` (*Warnings All*) care comandă compilatorul să afișeze mesaje de avertizare pentru abateri de la standard sau pentru formulări atipice:

```
mircea@cougar:~/carte-uso/cap-10$ gcc -Wall hw2.c -o my_exec
hw2.c:4: warning: return type defaults to 'int'
hw2.c: In function 'main': hw2.c:6: warning: control reaches end of non-
void function
```

După corectare, avertismentele nu vor mai fi furnizate:

```c
#include <stdio.h>

int main(void)
{
        printf("Hello, World!\n");

        return 0;
}
```

Listing 11.3: hw2.c

```
mircea@cougar:~/carte-uso/cap-10$ gcc -Wall hw2.c -o my_exec

mircea@cougar:~/carte-uso/cap-10$
```

O greșeală frecventă care poate fi eliminată prin folosirea opțiunii `-Wall` este utilizarea construcției `if (a = 1)`. Programul de mai jos folosește o astfel de construcție care este raportată, însă, de compilator în momentul folosirii opțiunii `-Wall`:

[1]Una dintre greșelile frecvente realizate de programatorii începători este folosirea formelor **void** `main(`**void**`)` sau `main()`. Standardul C specifică forma **int** `main(`**void**`)` ca fiind cea corectă. Nefolosirea acestei forme poate conduce la probleme - http://users.aber.ac.uk/auj/voidmain.shtml

```c
#include <stdio.h>

int main(void)
{
        int a = 1;

        if (a = 1)
                printf("Hello, World!\n");

        return 0;
}
```

Listing 11.4: hw3.c

```
mircea@cougar:~/carte-uso/cap-10$ gcc -Wall hw2.c -o my_exec
hw2.c: In function 'main': hw2.c:7: warning: suggest parentheses around
assignment used as truth value
```

IMPORTANT: Din cauza sintaxei relaxate a limbajului C este recomandată folosirea opțiunii `-Wall` la orice compilare de program.

11.2.2 Compilarea din surse multiple

O aplicație complexă va avea mai multe fișiere sursă conținând diverse părți din implementare. În exemplul de mai jos, fișierul `main.c` conține funcția `main`, fișierele `add.c` și `sub.c` conțin, respectiv, implementarea funcțiilor `add` și `sub`, iar fișierul `func.h` deține antetele (declarațiile) acelor funcții.

```c
#include <stdio.h>

#include "func.h"

#define NUM     10

int main(void)
{
        printf("Suma(%d) = %d\n", NUM, sum(NUM));
        printf("Suma-alternanta(%d) = %d\n", NUM, sumalt(NUM));

        return 0;
}
```

Listing 11.5: main.c

```c
#ifndef FUNC_H_
#define FUNC_H_         1

int sum(int n);
int sumalt(int n);

#endif
```

Listing 11.6: func.h

```c
#include "func.h"

```

```
3  int sum(int n)
4  {
5          int i, sum;
6
7          sum = 0;
8          for (i = 1; i <= n; i++)
9                  sum += i;
10
11         return sum;
12 }
```

Listing 11.7: sum.c

```
1  #include "func.h"
2
3  int sumalt(int n)
4  {
5          int i, sum;
6
7          sum = 0;
8          for (i = 1; i <= n; i++) {
9                  if (i % 2 == 0)
10                         sum -= i;
11                 else
12                         sum += i;
13         }
14
15         return sum;
16 }
```

Listing 11.8: sumalt.c

Pentru obținerea unui executabil, toate aceste fișiere vor fi transmise ca argument compilatorului:

```
1  razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ gcc -Wall main.c
   sum.c sumalt.c
2
3  razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ ./a.out
4  Suma(10) = 55
5  Suma-alternanta(10) = -5
```

Se poate folosi opțiunea `-o` pentru a specifica numele executabilului final:

```
1  razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ gcc -Wall main.c
   sum.c sumalt.c -o exec
2
3  razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ ./exec
4  Suma(10) = 55
5  Suma-alternanta(10) = -5
```

11.3 Etapele compilării (inclusiv link-editarea)

Exemplele de până acum au folosit GCC pentru a obține dintr-un fișier sau mai multe fișiere sursă un executabil. Acest lucru a fost realizat într-o singură comandă. Totuși, rularea unei instanțe a GCC presupune trecerea printr-un set de etape de prelucrare a fișierului sursă până la executabil. Aceste faze sunt, în ordine:

1. preprocesarea;
2. compilarea;
3. asamblarea;
4. link-editarea.

O rulare simplă a comenzii **gcc** înseamnă trecerea prin toate aceste faze. Prezența unor parametri poate forța GCC să se oprească după una dintre ele.

Figura 11.2 sintetizează etapele compilării:

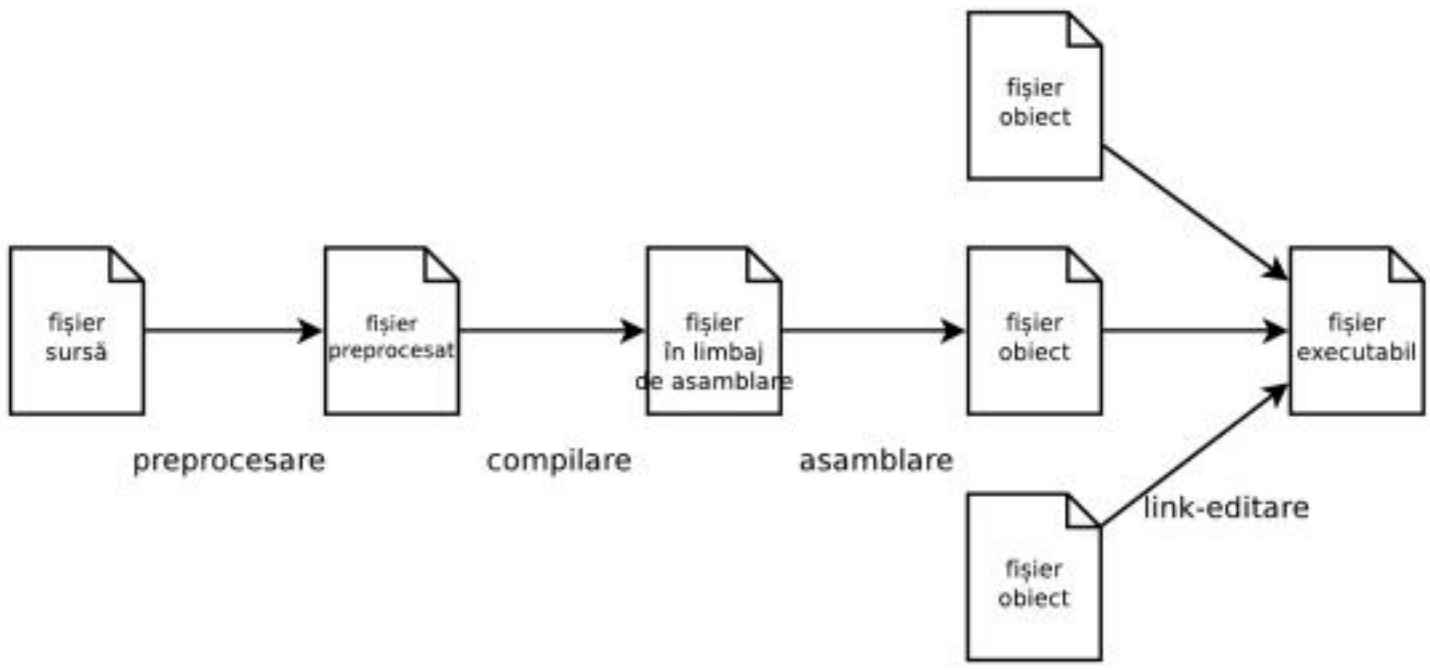


Figura 11.2: Fazele compilării

În cazul folosirii directe a comenzii **gcc**, se realizează integrat toate cele 4 faze. Fișierele obținute în urma fiecărei faze sunt, în această situație, stocate temporar în sistemul de fișiere după care sunt șterse. O rulare de tip *verbose* a comenzii **gcc** prezintă detaliat fazele procesului de compilare:

```
1  razvan@valhalla:~/code/ptr$ gcc -v -Wall ptr.c -o ptr
2  Using built-in specs.
3  Target: x86_64-linux-gnu
4  [...]
5   /usr/lib/gcc/x86_64-linux-gnu/4.3.3/cc1 -quiet -v ptr.c -quiet -dumpbase
   ptr.c -mtune=generic -auxbase ptr -Wall -version -o /tmp/ccYyQvdx.s
6  [...]
7  GNU C (Debian 4.3.3-14) version 4.3.3 (x86_64-linux-gnu) compiled by GNU
   C version 4.3.3, GMP version 4.3.1, MPFR version 2.4.1-p2.
8  [...]
9   as -V -Qy -o /tmp/ccoFhtN3.o /tmp/ccYyQvdx.s
10 GNU assembler version 2.19.51 (x86_64-linux-gnu) using BFD version (GNU
   Binutils for Debian) 2.19.51.20090723
11 [...]
12  /usr/lib/gcc/x86_64-linux-gnu/4.3.3/collect2 --build-id --eh-frame-hdr -
m elf_x86_64 --hash-style=both -dynamic-linker /lib64/ld-linux-x86-64.so
.2 -o ptr /usr/lib/gcc/x86_64-linux-gnu/4.3.3/../../../../lib/crt1.o
[...] /tmp/ccoFhtN3.o -lgcc --as-needed -lgcc_s --no-as-needed -lc -lgcc
[...]
```

După cum se observă mai sus, există 3 etape rezultate în urma rulării comenzii **gcc**:

1. invocarea compilatorului (**cc1**) conduce la obținerea fișierului în limbaj de asamblare `/tmp/ccYyQvdx.s`;

2. invocarea asamblorului (**as**) conduce la obținerea fișierului obiect `/tmp/ccoFhtN3.o`;
3. invocarea linker-ului (**collect2**) conduce la obținerea fișierului executabil `ptr`;

În cazul GCC, preprocesorul este invocat intern de compilator (**cc1**) și nu rezultă în obținerea unui fișier temporar.

În continuare vor fi prezentate mai detaliat fazele procesului de compilare și fișierele intermediare obținute.

11.3.1 Preprocesarea

Etapa de preprocesare presupune înlocuirea/expandarea directivelor de preprocesare din fișierul sursă.

Directivele de preprocesare încep cu `#` (*diez*, *sharp* în engleză). Printre acestea se numără directiva `#include` și directiva `#define`. După preprocesare, directiva `#include` este înlocuită cu fișierul inclus, iar directiva `#define` definește un macro care va fi substituit cu valoarea sa oriunde apare în codul sursă.

În exemplul de mai jos se realizează preprocesarea fișierului `info.c`.

```
1  #ifndef INFO_H_
2  #define INFO_H_          1
3
4  #define MAX_ARRAY_LENGTH        32
5
6  struct array_struct {
7          int array[MAX_ARRAY_LENGTH];
8          int len;
9  };
10
11 #endif
```

Listing 11.9: info.h

```
1  #include "info.h"
2
3  int get_sum(struct array_struct *a)
4  {
5          int i;
6          int sum;
7
8          if (a->len >= MAX_ARRAY_LENGTH)
9                  return 0;
10
11         for (i = 0; i < a->len; i++)
12                 sum += a->array[i];
13
14         return sum;
15 }
```

Listing 11.10: info.c

Pentru oprirea rulării GCC după faza de preprocesare se folosește opțiunea `-E`:

```
mircea@cougar:~/carte-uso/cap-10$ gcc -E info.c
# 1 "info.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "info.c"
# 1 "info.h" 1


struct array_strunct {
 int array[32];
 int len;
};
# 2 "info.c" 2

int get_sum(struct array_struct *a)
{
 int i;
 int sum;

 if (a->len >= 32)
  return 0;

 for (i = 0; i < a->len; i++)
  sum += a->array[i];

 return sum;
}
```

Se observă înlocuirea directivei `#include` cu fișierul `info.h` și înlocuirea macrodefiniției `MAX_ARRAY_LENGTH` cu `32` în fișierul preprocesat.

Preprocesarea este singura fază a procesului de compilare care nu rezultă în mod implicit într-un nou fișier. În absența opțiunii `-o`, rezultatul preprocesării este scris la ieșirea standard.

Folosirea opțiunii `-o` sau a operatorului de redirectare permite precizarea fișierului de ieșire. Convențional, fișierul preprocesat are extensia `.i`:

```
mircea@cougar:~/carte-uso/cap-10$ gcc -E info.c -o info.i

mircea@cougar:~/carte-uso/cap-10$ gcc -E info.c > info.i
```

Preprocesarea este efectuată pe un sistem Linux cu ajutorul utilitarului GNU CPP (C preprocessor)[1].

11.3.2 Compilarea

> **Compilarea** este etapa în care din fișierul preprocesat se obține un fișier în limbaj de asamblare.

Compilarea efectivă înseamnă traducerea codului din limbaj de nivel înalt în limbaj de asamblare. Limbajul de asamblare descrie instrucțiunile specifice procesorului

[1] http://gcc.gnu.org/onlinedocs/cpp/

sistemului fizic. Pentru oprirea rulării GCC după faza de compilare se folosește opțiunea `-S`.

În exemplul de mai jos, se compilează fișierul `info.c` din secțiunea anterioară și se obține, implicit, fișierul `info.s`. Se poate preciza explicit fișierul de ieșire cu ajutorul opțiunii `-o`.

```
mircea@cougar:~/carte-uso/cap-10$ ls info.*
info.c  info.h

mircea@cougar:~/carte-uso/cap-10$ gcc -S info.c

mircea@cougar:~/carte-uso/cap-10$ ls info.*
info.c  info.h  info.s
```

```
        .file   "info.c"
        .text
.globl get_sum
        .type   get_sum, @function
get_sum:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $20, %esp
        movl    8(%ebp), %eax
        movl    128(%eax), %eax
        cmpl    $31, %eax
        jle     .L2
        movl    $0, -20(%ebp)
        jmp     .L4
[...]
```

Listing 11.11: info.s

Pe sistemele Linux și Unix, fișierul în limbaj de asamblare are, de obicei, extensia `.s`. Pe sistemele Windows, un fișier în limbaj de asamblare are extensia `.asm`.

Limbaje de asamblare

Limbajul de asamblare este un limbaj de programare de nivel scăzut. Instrucțiunile sale (denumite și mnemonici) sunt de fapt reprezentarea simbolică a instrucțiunilor puse la dispoziție de procesorul sistemului. Exemple de mnemonici sunt `mov`, `cmp`, `add`, `sub`, `push` etc. Instrucțiunea în cod mașină:

```
10110000 01100001 (0xb061)
```

este scrisă în limbaj de asamblare ca:

```
mov     al,     061h
```

Mnemonica `mov` înseamnă efectuarea unei copieri. Rezultatul este încărcarea valorii `0x61` în registrul `al` al procesorului. Există, în general, o asociere unu la unu între o instrucțiune în limbaj de asamblare și o instrucțiune în cod mașină.

Un program scris în limbaj de asamblare este neportabil: poate fi rulat numai pe arhitectura de calcul pentru care a fost scris. Dacă programul în limbaj de asamblare trebuie portat pe o altă arhitectură, trebuie rescris și folosite mnemonicile spcifice acelei arhitecturi.

Întrucât necesită cunoștințe despre structura procesorului, resursele acestuia, mecanismul de acces la memorie, programarea în limbaj de asamblare este dificilă. Se recomandă folosirea acestuia doar la nevoie:

- când un limbaj cum este C nu oferă instrucțiuni pentru efectuarea unei anumite operații
- când este nevoie de eficiență maximă care poate fi obținută numai prin scrierea de cod în limbaj de asamblare

Traducerea codului în limbaj de asamblare în cod obiect se realizează cu ajutorul unui asamblor în faza de asamblare descrisă în secțiunea următoare. Fiind vorba de o asociere unu la unu între mnemonici și instrucțiunile procesorului, este posibilă obținerea codului în limbaj de asamblare pornind de la codul obiect. Această operație poartă numele de dezasamblare și este prezentată în secțiunea 11.3.3.

Tipuri de limbaje de asamblare

Chiar pentru același procesor pot exista limbaje de asamblare diferite sau, mai corect spus, sintaxe de limbaj de asamblare diferite. Pe arhitectura x86, asamblorul de la Microsoft folosește sintaxa Intel, pe când asamblorul de Linux (GNU) folosește sintaxa AT&T[1]. Astfel, copierea valorii `0x100` în registrul `eax` se face, folosind cele două sintaxe, astfel:

```
1 movl   $0x100,  %eax    ; sintaxa AT&T
2 mov    eax,     100h    ; sintaxa Intel
```

La fel, stocarea valorii `100` la adresa indicată de `eax`:

```
1 movl $0x100,    (eax)   ; sintaxa AT&T
2 movl [eax],     100h    ; sintaxa Intel
```

11.3.3 Asamblarea

Etapa de asamblare este etapa de traducere a codului scris în limbaj de asamblare (fișier text conținând mnemonici specifice arhictecturii sistemului de calcul) în cod binar. Acest cod binar este cod mașină reprezentând codificarea binară a instrucțiunilor procesorului. Fișierul obținut poartă numele de fișier cod obiect sau modul obiect. Pe Unix un modul obiect are extensia `.o` iar pe Windows extensia `.obj`.

Pentru oprirea rulării **gcc** după faza de asamblare se folosește opțiunea `-c`. În mod implicit se înlocuiește extensia fișierului sursă cu `.o`. Fișierul sursă poate fi un fișier C sau un fișier în limbaj de asamblare:

```
1 mircea@cougar:~/carte-uso/cap-10$ gcc -c info.c
2
3 mircea@cougar:~/carte-uso/cap-10$ ls -l info.o
4 -rw-r--r-- 1 ubuntu ubuntu 764 Sep 19 00:29 info.o
5
6 mircea@cougar:~/carte-uso/cap-10$ gcc -c info.s
7
```

[1] http://en.wikipedia.org/wiki/X86_assembly_language

```
mircea@cougar:~/carte-uso/cap-10$ ls -l
info.o -rw-r--r-- 1 ubuntu ubuntu 744 Sep 19 00:29 info.o
```

Ca și până acum, se poate folosi opțiunea `-o` pentru a preciza explicit numele fișierului de ieșire:

```
mircea@cougar:~/carte-uso/cap-10$ gcc -c info.s -o modul.o

mircea@cougar:~/carte-uso/cap-10$ ls -l modul.o
-rw-r--r-- 1 ubuntu ubuntu 744 Sep 19 00:47 modul.o
```

În cadrul suitei de aplicații GNU, asamblorul folosit Gas (GNU Assembler)[1], iar executabilul asociat este `as`. Un fișier în limbaj de asamblare poate fi asamblat folosind `as`:

```
mircea@cougar:~/carte-uso/cap-10$ as info.s -o as_out.o

mircea@cougar:~/carte-uso/cap-10$ ls -l as_out.o
-rw-r--r-- 1 ubuntu ubuntu 744 Sep 19 00:48 as_out.o
```

Modul obiect

Un modul obiect conține cod mașină specific procesorului. Acest cod este obținut în urma fazelor de preprocesare, compilare și asamblare. Un modul obiect are, de obicei, extensia `.o`:

```
mircea@cougar:~/carte-uso/cap-10$ file info.o
info.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not
stripped
```

objdump; dezasamblare

Datorită asocierii unu la unu care există între codul obiect și codul în limbaj de asamblare, se poate realiza o operație de dezasamblare.

> **Dezasamblare** înseamnă "recuperarea" mnemonicilor în limbaj de asamblare pornind de la codul mașină; este operația inversă asamblării.

Utilitarul folosit în acest sens este **`objdump`**:

```
razvan@anaconda:~/uso/compilare$ objdump -d info.o
info.o:     file format elf32-i386

Disassembly of section .text:

00000000 <get_sum>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 ec 14                sub    $0x14,%esp
   6:   8b 45 08                mov    0x8(%ebp),%eax
   9:   8b 80 80 00 00 00       mov    0x80(%eax),%eax
   f:   83 f8 1f                cmp    $0x1f,%eax
```

[1] http://en.wikipedia.org/wiki/GNU_Assembler

```
  12:   7e 09                   jle    ld <get_sum+0x1d>
  14:   c7 45 ec 00 00 00 00    movl   $0x0,0xffffffec(%ebp)
[...]
```

nm

Un alt utilitar folositor în lucrul cu fișiere obiect este **nm**. Cu ajutorul acestuia se pot lista simbolurile dintr-un fișier obiect:

```
razvan@anaconda:~/uso/compilare$ nm info.o
00000000 T get_sum
```

nm poate oferi informații despre definirea sau nu a unui simbol într-un modul obiect:

```
razvan@anaconda:~/uso/compilare$ nm main.o
         U add
00000000 T main
         U printf
         U sub
```

Din rezultatul afișării se observă că simbolurile sunt marcate cu `U` sau `T`. `T` înseamnă că acel simbol este definit în zona de text (cod) a executabilului, iar `U` înseamnă că este nedefinit în modulul obiect primit ca argument. În exemplul nostru, modulul are definit simbolul `main` dar nedefinite simbolurile `add`, `sub` și `printf`. Aceste simboluri se găsesc în alte module obiect sau în biblioteci. Rezolvarea acestor simboluri se va realiza în momentul link-editării.

11.3.4 Optimizarea compilării

Un compilator translatează codul sursă în limbaj de asamblare, și după faza de asamblare, în cod mașină. Operațiile descrise într-un limbaj de nivel înalt pot fi traduse în diverse "scenarii" de instrucțiuni în limbaj de asamblare. Având cunoștințe intime despre arhitectura sistemului de calcul (setul de instrucțiuni, registrele procesorului și modurile de adresare), compilatorul poate alege un scenariu mai bun de translatare.

> Faza translatării în care compilatorul încearcă obținerea unui cod mașină mai bun poartă numele de **optimizarea codului**.

Denumirea de *optimizare* este ușor forțată. Un termen mult mai bun este cel de *îmbunătățire*; nu se poate spune că se obține *cod optim*. Datorită răspândirii acestei forme, atât în limba română cât și în limba engleză (*optimizing compiler*), se va folosi în continuare denumirea de *optimizare*.

Optimizarea are drept scop obținerea unui executabil care să ruleze cât mai rapid și/sau care să ocupe cât mai puțin spațiu.

Se poate întâmpla ca aceste două scopuri (viteză mare, spațiu ocupat mic) să fie în conflict; utilizatorul va putea preciza compilatorului care scop este mai important.

O dată cu dezvoltarea sistemelor embedded, un rol important pe care trebuie să-l îndeplinească executabilul obținut de compilator este consumul redus de putere.

Folosirea optimizării reprezintă un compromis între obținerea unui executabil mai rapid sau care ocupă mai puțin spațiu și un timp mai îndelungat de compilare. Din acest motiv, partea de dezvoltare a unei aplicații se realizează fără optimizare. Faza de optimizare este folosită în momentul în care aplicația este pregătită pentru lansare (release).

Pentru a justifica avantajele și dezavantajele folosirii optimizării, mai jos se prezintă două scenarii de compilare a unei aplicații C++:

- fără optimizare; se obține executabilul `bb-ssa-no-opt`;
- cu folosirea unor opțiuni de optimizare; se obține executabilul `bb-ssa-opt`.

Se poate observa, ca *dezavantaj* o creștere a timpului necesar pentru compilare (de la `4.1` secunde la `5.7` secunde), dar ca *avantaj* o scădere a dimensiunii executabilului (de la `328KB` la `140KB`):

```
1  razvan@valhalla:~/school/2001-2006_code/pt/tema4/pt4$ time g++ -o bb-ssa-
   no-opt bb-ssa.cpp basic_block.cpp ssa.cpp asamblor.c analizor.c
   interpretor.tab.c -lfl
2
3  real    0m4.143s
4  user    0m3.624s
5  sys     0m0.488s
6
7  razvan@valhalla:~/school/2001-2006_code/pt/tema4/pt4$ time g++ -O -o bb-
   ssa-opt bb-ssa.cpp basic_block.cpp ssa.cpp asamblor.c analizor.c
   interpretor.tab.c -lfl
8
9  real    0m5.769s
10 user    0m5.288s
11 sys     0m0.456s
12
13 razvan@valhalla:~/school/2001-2006_code/pt/tema4/pt4$ ls -lh bb-ssa-no-
   opt
14 -rwxr-xr-x 1 razvan razvan 328K Aug 21 22:11 bb-ssa-no-opt
15
16 razvan@valhalla:~/school/2001-2006_code/pt/tema4/pt4$ ls -lh bb-ssa-opt
17 -rwxr-xr-x 1 razvan razvan 140K Aug 21 22:12 bb-ssa-opt
```

GCC permite optimizarea codului obținut prin intermediul opțiunii `-O`. Aceasta este urmată de un număr care specifică gradul de optimizare/îmbunătățire a codului. În mod implicit, compilatorul nu optimizează codul. Acest lucru este echivalent cu folosirea opțiunii `-O0` (litera `o` majusculă și `zero`):

```
1  mircea@cougar:~/carte-uso/cap-10$ gcc -c chatserv.c -O0
2
3  mircea@cougar:~/carte-uso/cap-10$ ls -l
4  chatserv.o -rw-r--r-- 1 ubuntu ubuntu 3656 Sep 18 23:46 chatserv.o
```

GCC prezintă 3 niveluri numerice de compilare:

- opțiunea `-O1` înseamnă reducerea timpului de execuție și a codului obținut fără a afecta semnificativ timpul de compilare; este echivalentă cu folosirea opțiunii `-O` (fără un argument numeric)

```
1  mircea@cougar:~/carte-uso/cap-10$ gcc -c chatserv.c -O1
2
3  mircea@cougar:~/carte-uso/cap-10$ ls -l
4  chatserv.o -rw-r--r-- 1 ubuntu ubuntu 3200 Sep 18 23:46 chatserv.o
```

- opțiunea `-O2` înseamnă folosirea tuturor optimizărilor care nu atrag un compromis viteza-spațiu

```
mircea@cougar:~/carte-uso/cap-10$ gcc -c chatserv.c -O2

mircea@cougar:~/carte-uso/cap-10$ ls -l
chatserv.o -rw-r--r-- 1 ubuntu ubuntu 3232 Sep 18 23:47 chatserv.o
```

Se observă că s-a obtinut un cod ceva mai mare decât în cazul opțiunii `-O1`, dar care va rula mai rapid. Nu întotdeauna un cod mai mic va fi mai rapid.

- opțiunea `-O3` este în acest moment cea mai puternică (și agresivă) metodă de compilare:

```
mircea@cougar:~/carte-uso/cap-10$ gcc -c chatserv.c -O3

mircea@cougar:~/carte-uso/cap-10$ ls -l
chatserv.o -rw-r--r-- 1 ubuntu ubuntu 3308 Sep 18 23:47 chatserv.o
```

- GCC oferă și opțiunea `-Os` care înseamnă optimizarea codului pentru a ocupa cât mai puțin spațiu. O utilizare tipică este în cazul dispozitivelor embedded care dispun de resurse hardware reduse:

```
mircea@cougar:~/carte-uso/cap-10$ gcc -c chatserv.c -Os

mircea@cougar:~/carte-uso/cap-10$ ls -l
chatserv.o -rw-r--r-- 1 ubuntu ubuntu 2868 Sep 18 23:47 chatserv.o
```

11.3.5 Link-editarea

Un modul obiect este asociat unui singur fișier/modul sursă. Modulul obiect descrie datele și codul funcțiilor proprii. Varibialele și funcțiile poartă numele generic de simboluri. Simbolurile externe modulului (adică funcțiile sau variabilelele externe – nedefinite local, prezente în alte module) sunt marcate ca nedefinite în cadrul modulului.

Pentru obținerea unui fișier executabil și deci pentru rularea acestui cod este necesară:

- identificarea simbolurilor nedefinite, operație denumită și rezolvarea simbolurilor, și
- unificarea sau legarea (linking) a zonelor de date și cod asociate într-un fișier executabil.

Această operație se numește link-editare, linking sau legare.

Spre exemplu, în momentul în care se compilează un modul obiect care folosește funcția de bibliotecă `printf`, simbolul apare ca fiind nedefinit (caracterul `U` la **nm**):

```
razvan@anaconda:~/uso/compilare$ nm main.o
00000000 T main
         U printf
```

Funcția `printf` este definită în biblioteca standard C. În urma link-editării se obține un executabil care importă zona de cod asociată funcției `printf` din biblioteca standard C. Altă situație este folosirea într-un modul obiect a unei funcții definite într-un alt modul

obiect; un exemplu este folosirea funcției `sum` în fișierul `main.c` descris în secțiunea 11.2.2. Funcția `sum` este definită în `sum.c`.

> Aplicația care rezolvă simbolurile nedefinite și care realizează legarea modulelor obiect poartă numele de **linker**.

Rolul unui linker este, așadar, de a lega mai multe fișiere obiect sau biblioteci, de a rezolva simbolurile nedefinite și de a obține un executabil sau o bibliotecă.

Figura 11.3 prezintă rolul îndeplinit de un linker.

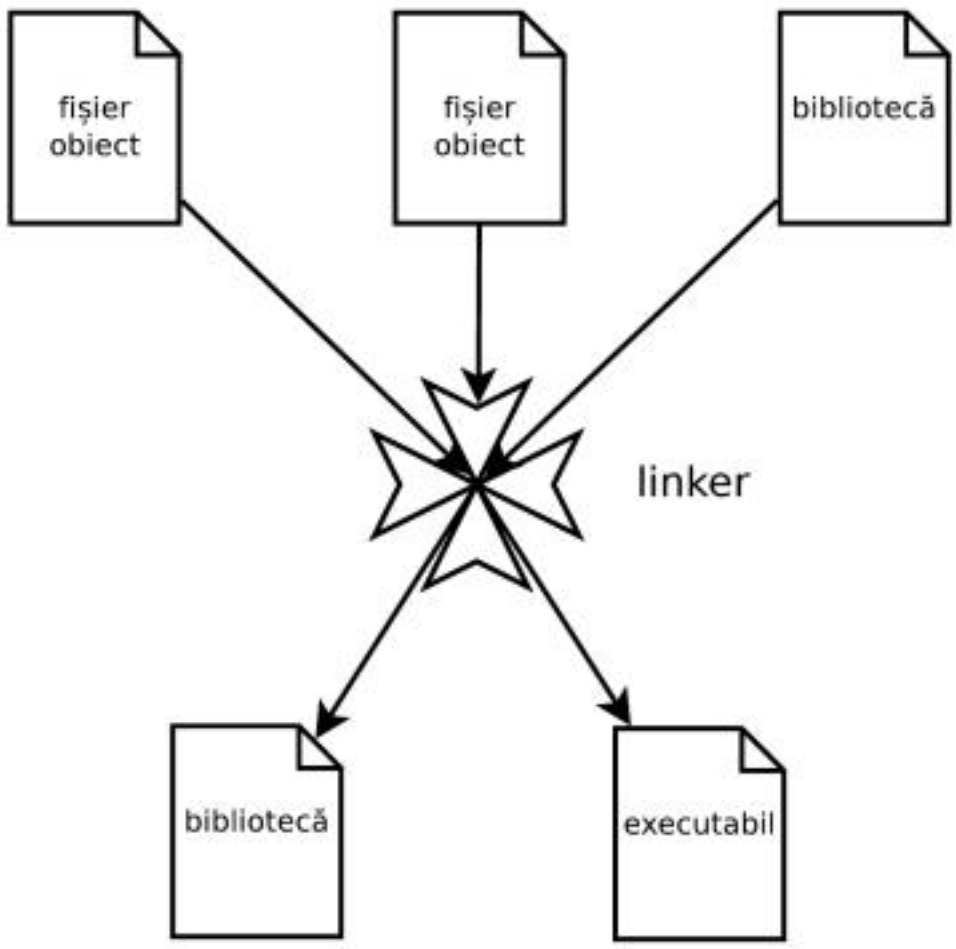


Figura 11.3: Funcționarea linker-ului

Mai multe detalii despre biblioteci se găsesc în secțiunea 11.4.

Invocare linker

Linker-ul de pe distribuțiile GNU/Linux este GNU linker sau GNU ld[1]. Executabilul asociat este **`ld`**. Linker-ul este rareori invocat cu ajutorul comenzii **`ld`**. În mod obișnuit, linker-ul este invocata intern de comanda **`gcc`**:

```
1  mircea@cougar:~/carte-uso/cap-10$ gcc -Wall -c main.c
2
3  mircea@cougar:~/carte-uso/cap-10$ gcc -Wall -c add.c
4
5  mircea@cougar:~/carte-uso/cap-10$ gcc -Wall -c sub.c
6
7  mircea@cougar:~/carte-uso/cap-10$ gcc main.o add.o sub.o -o exec
8
9  mircea@cougar:~/carte-uso/cap-10$ ./exec
10 Suma (10): 55
```

[1] http://en.wikipedia.org/wiki/GNU_linker

```
11 Suma alternanta (10): -5
```

Se poate folosi și **ld**, însă numărul mare de argumente care trebuie transmis face această opțiune mai puțin viabilă:

```
1 mircea@cougar:~/carte-uso/cap-10$ ld -dynamic-linker /lib/ld-linux.so.2 /
  usr/lib/crt1.o /usr/lib/crti.o /usr/lib/gcc/i486-linux-gnu/4.1.2/crtbegin
  .o main.o add.o sub.o -lc /usr/lib/gcc/i486-linux-gnu/4.1.2/crtend.o /usr
  /lib/crtn.o -o exec
2
3 mircea@cougar:~/carte-uso/cap-10$ ./exec
4 Suma (10): 55
5 Suma alternanta (10): -5
```

11.3.6 Fișiere executabile

> Un **fișier executabil** este un fișier binar obținut dintr-un set de fișiere obiect și biblioteci în urma operației de link-editare.

Spre deosebire de fișierele obiect, un fișier executabil are identificate și, în general, rezolvate simbolurile. Aceste simboluri sunt rezolvate la linking (legare statică) sau la rulare (legare dinamică).

În momentul lansării în execuție, informațiile conținute în fișierul executabil sunt folosite pentru a genera un proces. Un fișier executabil va conține codul mașină folosit pentru îndeplinirea sarcinilor date dar și antete și secțiuni de formatare auxiliare. Acestea sunt folosite pentru a descrie modul în care se folosește codul, zonele de memorie folosite etc.

Un fișier executabil are, așadar, un format bine definit. Formatul de fișier executabil se referă la fișiere obiect, fișiere executabile și la biblioteci cu legare dinamică.

Formate ale fișierelor executabile

Formatul unui fișier executabil[1] este strâns legat de sistemul de operare. Sistemul de operare este responsabil cu interpretarea unui fișier executabil și generarea unui proces pe baza acestuia. Un executabil poate fi rulat pe un sistem de operare dacă acel sistem de operare oferă suport pentru formatul de executabil folosit. Formatul de fișier executabil se referă și la fișierele obiect.

Exemple de formate de fișiere obiect/executabile sunt:

- **ELF** – folosit în sistemele Unix;
- **COFF** – versiunea anterioară ELF folosită pe Unix;
- **a.out** – primul format folosit de sistemele Unix; a oferit denumirea pentru executabilul implicit generat de compilator/linker;
- **PE** – formatul implicit pe sistemele Windows;

[1] http://en.wikipedia.org/wiki/Object_file

- **Mach-O** – formatul implicit pe Mac OS X.

În Linux, formatul de fișier obiect/executabil folosit este ELF (Executable and Linking Format)[1]:

```
1  mircea@cougar:~/carte-uso/cap-10$ file info.o
2  info.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not
   stripped
3
4  mircea@cougar:~/carte-uso/cap-10$ file exec
5  exec: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/
   Linux 2.4.1, dynamically linked (uses shared libs), for GNU/Linux 2.4.1,
   not stripped
6
7  mircea@cougar:~/carte-uso/cap-10$ file /bin/ls
8  /bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for
   GNU/Linux 2.4.1, dynamically linked (uses shared libs), for GNU/Linux
   2.4.1, stripped
9
10 mircea@cougar:~/carte-uso/cap-10$ file /lib/libc-2.3.6.so
11 /lib/libc-2.3.6.so: ELF 32-bit LSB shared object, Intel 80386, version 1
   (SYSV), for GNU/Linux 2.4.1, stripped
```

În listingul de mai sus se observă tipurile de fișiere care folosesc formatul ELF:

- module obiect; fișierul `info.o` este un modul obiect ELF relocabil[2];
- fișiere executabile; fișierele `exec` și `/bin/ls` sunt fișiere executabile format ELF;
- biblioteci partajate (vezi x.y.z); fișierul `/lib/libc-2.3.6.so` este o bibliotecă partajată (shared-object) (vezi secțiunea 11.4.1).

Utilitarul **`readelf`** poate fi folosit pentru analiza unui fișier în format ELF. În exemplul de mai jos se pot observa câteva din secțiunile unui fișier executabil: `.text` reprezintă zona de cod, `.rodata` zona de date read-only.

```
1  razvan@anaconda:~/uso/compilare$ readelf -S exec
2  There are 34 section headers, starting at offset 0xe38:
3
4  Section Headers:
5    [Nr] Name              Type            Addr     Off    Size   ES Flg Lk
    Inf Al
6    [ 0]                   NULL            00000000 000000 000000 00      0
     0  0
7    [ 1] .interp           PROGBITS        08048114 000114 000013 00   A  0
     0  1
8    [ 2] .note.ABI-tag     NOTE            08048128 000128 000020 00   A  0
     0  4
9    [ 3] .hash             HASH            08048148 000148 000028 04   A  4
     0  4
10 [...]
```

[1] http://en.wikipedia.org/wiki/Executable_and_Linkable_Format
[2] http://en.wikipedia.org/wiki/Relocation_(computer_science)

11.4 Biblioteci de funcții

O **bibliotecă** este colecție de funcții sau clase care oferă servicii preimplementate dezvoltatorului. În general, o bibliotecă este obținută prin comasarea mai multor fișiere obiect. O bibliotecă este, de asemenea, un fișier.

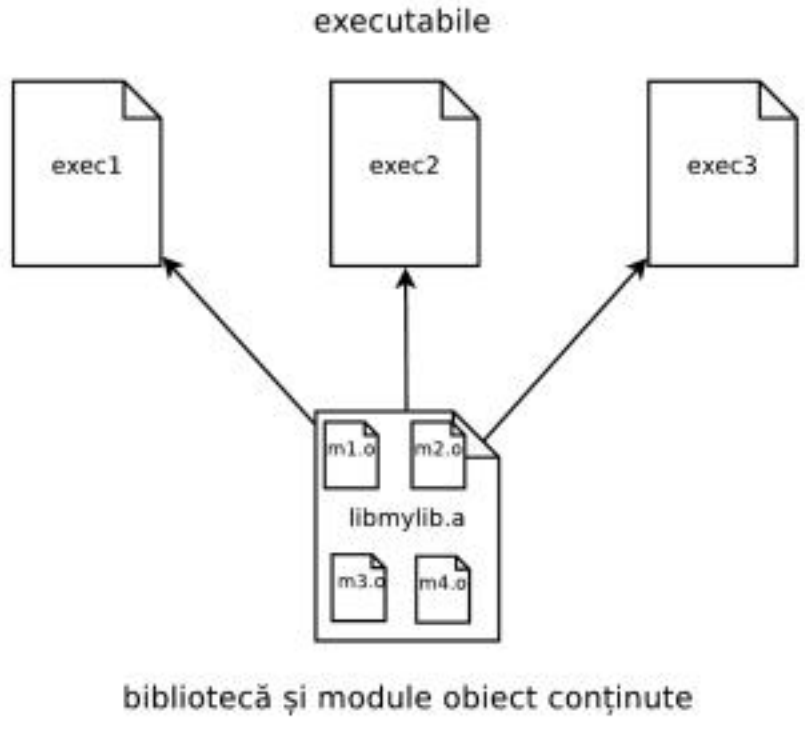


Figura 11.4: Bibliotecă

Bibliotecile de funcții (engleză: *libraries*) au aparut pentru a oferi o mai mare flexibilitate dezvoltatorilor de software. Acestea reunesc funcții des utilizate, astfel încât un program va folosi o bibliotecă fără a fi nevoie de rescrierea codului dorit.

În zilele noastre, foarte multe servicii au fost încapsulate în biblioteci de funcții pentru a ușura procesul de dezvoltare de noi aplicații. Astfel:

- biblioteca standard C (`libc`) oferă serviciile de bază ale sistemului;
- biblioteci precum GTK+[1] sau Qt[2] oferă primitive pentru dezvoltarea de interfețe grafice;
- utilitarul Pidgin[3] de instant messaging a izolat funcțiile principale în biblioteca `libpurple`;
- dezvotatorii Subversion[4] au dezvoltat biblioteca `libsvn1`;
- `libpng`, `libjpeg`, `libtiff` sunt folosite pentru prelucrarea de imagini;
- `libtorrent-rakshasa` și `libtorrent-rasterbar` sunt biblioteci care implementează protocolul BitTorrent folosite de un număr mare de clienți BitTorrent.

Mai sus sunt prezentate doar câteva dintre aplicațiile bibliotecilor. Practic, orice funcționalitate utilizabilă de mai multe aplicații poate fi implementată într-o bibliotecă.

[1] http://www.gtk.org/
[2] http://qt.nokia.com/
[3] http://www.pidgin.im/
[4] http://subversion.tigris.org/

Din circa 25000 de pachete instalabile în cadrul distribuției Debian GNU/Linux Testing/Squeeze[1], peste 5000 sunt biblioteci:

```
1 razvan@valhalla:~/code/ptr$ apt-cache search library | grep ^lib | wc -l
2 5699
```

11.4.1 Tipuri de biblioteci

Bibliotecile își încep utilizarea în momentul link-editării. Pentru rezolvarea simbolurilor nedefinite, linker-ul caută în modulele obiect încapsulate în cadrul bibliotecilor și realizează rezolvarea simbolurilor. Rezolvarea înseamnă marcarea locului în care simbolul este definit. În funcție de tipul bibliotecii, linker-ul decide să adauge modulul obiect necesar în codul executabilului sau doar să marcheze referința rezolvată. Există, astfel, două tipuri de biblioteci:

- **biblioteci cu legare statică** (*statically-linked libraries*) sau, pe scurt, *biblioteci statice*;
- **biblioteci cu legare dinamică** (*dinamically-linked libraries*) denumite *biblioteci partajate* (*shared libraries*) pe Linux și *dynamic-link libraries* (DLL) pe Windows.

Indiferent de tip, bibliotecile de funcții există sub formă de fișiere independente în sistem.

Bibliotecile statice (static libraries) sunt acele biblioteci ale căror module obiect componente sunt incluse în fișierul executabil în momentul link-editării.

Altfel spus, un executabil obținut în urma legării cu o bibliotecă statică deține tot codul necesar pentru a rula. Dimensiunea executabilului este mărită prin includerea fișierelor obiect necesare din cadrul bibiotecii.

În cazul **bibliotecilor partajate (shared libraries)**, operația de link-editare doar marchează referințele ca fiind rezolvate. Modulele obiect nu sunt incluse în codul executabilului obținut; vor fi adăugate în momentul lansării în execuție sau în momentul rulării.

Codul asociat unei biblioteci partajate nu ajunge niciodată în cadrul unui executabil. Biblioteca va fi încărcată în memorie la nevoie și va fi folosită în cadrul procesului obținut în urma execuției. În funcție de momentul în care biblioteca va fi încărcată în memorie, există două subtipuri de biblioteci cu legare dinamică:

- **biblioteci cu încărcare la execuție** (*load-time dynamically-linked library*) – biblioteca este adusă în memorie în momentul în care programul executat, dacă nu există deja acolo;
- **biblioteci cu încărcare la rulare** (*run-time dinamically-linked library*) – biblioteca este adusă în memorie la cerere, în momentul în care programul execută o instrucțiune care solicită acest lucru.

Bibliotecile partajate, după cum le spune și denumirea, reprezintă metoda cea mai utilizată de a pune la dispoziție funcții comune pentru mai multe aplicații. Dacă două

[1] http://packages.debian.org/testing/

sau mai multe aplicații folosesc aceeași bibliotecă, aceasta va trebui să fie încărcată o singură dată în memorie, economisind astfel spațiul ocupat. Biblioteca standard C, folosită de cea mai mare parte a aplicațiilor, este, de obicei, o bibiotecă partajată.

Tabela 11.1 reprezintă o comparație între bibliotecile statice și cele partajate:

Tabelul 11.1: Comparație între tipurile de biblioteci

Biblioteci statice	**Biblioteci partajate**
Colecție/arhivă de module obiect	Fișier format ELF
Executabil rezultat mai mare; se include cod	Executabil de dimensiune mică; nu se include cod
Executabilul poate fi mutat pe alt sistem	Executabilul are nevoie de prezența bibliotecii pentru a putea rula
Codul necesar este adăugat fiecărui executabil și apoi fiecărui proces	Codul bibliotecii este partajat între toate procesele care o folosesc
Timp de execuție mai rapid	Timp de execuție mai lent; se face rezolvarea simbolurilor în mod dinamic

11.4.2 Informații despre bibliotecile de funcții

Pe un sistem Linux, bibliotecile se găsesc în directoarele `/lib`, `/usr/lib` sau `/usr/local/lib`. Bibliotecile statice au extensia `.a`, iar cele partajate `.so` (de la shared object).

Informații primare despre bibliotecile de funcții se pot afla prin intermediul comenzii **`file`**:

```
1 mircea@cougar:~/carte-uso/cap-10$ file /usr/lib/libm.a
2 /usr/lib/libm.a: current ar archive
3
4 mircea@cougar:~/carte-uso/cap-10$ file /lib/libm.so.6
5 /lib/libm.so.6: symbolic link to 'libm-2.3.3.so'
6
7 mircea@cougar:~/carte-uso/cap-10$ file /lib/libm-2.3.3.so
8 /lib/libm-2.3.3.so: ELF 32-bit LSB shared
9 object, Intel 80386, version 1 (SYSV), not stripped
```

Bibliotecile statice sunt de fapt arhive de fișiere obiect. Ele sunt create prin intermediul comenzii **`ar`**. (pentru mai multe informații se poate consulta man **`ar`**).

O bibliotecă partajată este creată de linker. Pe un sistem Linux, o bibliotecă partajată se creează cu ajutorul comenzii **`gcc`** și opțiunea `-shared`[1]. O bibliotecă partajată este identificată printr-un număr de versiune. Pentru a facilita folosirea acesteia, se creează o legătură care nu include numărul versiunii.

Comanda **`nm`** poate fi utilizată pentru a vizualiza lista simbolurilor dintr-o bibliotecă (statică sau dinamică):

```
1 mircea@cougar:~/carte-uso/cap-10$ nm /lib/libm.so.6
2 00052b60 T fscanf
3 00043f20 T fprintf
```

[1] http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html

```
00052b90 T scanf
00052bd0 T sscanf
         U __libc_enable_secure@@GLIBC_PRIVATE
         U __libc_stack_end@@GLIBC_2.1
[...]
```

Simbolurile marcate cu `T` sunt simboluri din zona de cod (funcții) ce pot fi folosite extern de alte programe sau alte biblioteci. Simbolurile care sunt precedate de caracterul `U` sunt simboluri care trebuie rezolvate de linker la apelarea programului (se observă că ele nu au adresă în fișierul cu biblioteca). Acestea nu sunt definite local, ci în alte biblioteci (simbolul `__libc_enable_secure@@GLIBC_PRIVATE` este definit în biblioteca standard C – libc)

11.4.3 Utilizarea bibliotecilor

Bibliotecile reprezintă colecții de module preimplementate folosite pentru obținerea unui executabil, ceea ce înseamnă că, în momentul link-editării, linker-ului îi trebuie precizate bibliotecile necesare. Acest lucru se realizează cu ajutorul opțiunii `-l` (litera "L mic"). În listingul 11.4.3 se apelează funcții din biblioteca `ncurses`.

```c
#include <stdio.h>
#include <stdlib.h>
#include <ncurses.h>

int main(void)
{
        initscr();
        cbreak(); noecho();
        printf("Smash␣forehead␣on␣keyboard␣to␣continue␣...");
        fflush(stdout); fflush(stdin);
        getch();
        nocbreak(); echo();
        endwin();

        return 0;
}
```

Listing 11.12: Folosire funcții din biblioteca ncurses

În urma comenzii de compilare și rulare rezultă următoarea eroare:

```
razvan@valhalla:~/carte-uso/cap-10$ gcc -Wall -o getch getch.c
/tmp/ccavsseB.o: In function 'main':
getch.c:(.text+0x5): undefined reference to 'initscr'
getch.c:(.text+0xa): undefined reference to 'cbreak'
getch.c:(.text+0xf): undefined reference to 'noecho'
getch.c:(.text+0x3d): undefined reference to 'stdscr'
getch.c:(.text+0x42): undefined reference to 'wgetch'
getch.c:(.text+0x47): undefined reference to 'nocbreak'
getch.c:(.text+0x4c): undefined reference to 'echo'
getch.c:(.text+0x51): undefined reference to 'endwin'
collect2: ld returned 1 exit status
```

Linker-ul nu găsește definițiile funcțiilor `initscr`, `cbreak`, `noecho` etc. Pentru a rezolva această problemă link-erului îi trebuie specificat să folosească biblioteca `ncurses`. Aceasta este localizată în fișierul `libncurses.a` sau `libncurses.so`:

```
razvan@valhalla:~/carte-uso/cap-10$ ls /usr/lib/libncurses.*
/usr/lib/libncurses.a  /usr/lib/libncurses.so  /usr/lib/libncurses.so.5
```

Numele sub care va fi folosită biblioteca de linker se obține prim eliminararea extensiei (`.a` sau `.so`) și a prefixului `lib`. Pentru face legarea fișierului `getch.c` cu biblioteca `ncurses`, se folosește comanda:

```
razvan@valhalla:~/carte-uso/cap-10$ gcc -Wall -o getch \
> getch.c -lncurses

razvan@valhalla:~/carte-uso/cap-10$ ls -l getch
-rwxr-xr-x 1 razvan razvan 10734 Sep 21 23:25 getch
```

Se observă că, în acest caz, operația de linking se desfășoară cu sucess și se obține executabilul `getch`.

Bibliotecile sunt căutate în zone standard din sistemul de fișiere: `/lib`, `/usr/lib`, `/usr/local/lib`. În cazul în care fișierul asociat bibliotecii nu se găsește într-unul din directorele standard, se folosește opțiunea `-L`. Astfel, dacă se dorește legarea cu o biblioteca definită în fișierul `/home/traian/libs/libavatar.a`, se folosește o comandă de forma:

```
ubuntu@ubuntu:~$ gcc -Wall -o morph morph.c -lavatar -L/home/traian/libs
```


Când se specifică un director suplimentar în care se va realiza căutarea, numele acestuia se "lipește" de opțiunea `-L`; **nu** se lasă spații libere între opțiunea `-L` și numele directorului.

În cazul bibliotecilor partajate, există o etapă suplimentară de căutare a bibliotecii în momentul lansării în execuție a programului (la încărcare). Modul în care se specifică directoare suplimentare de căutare depășește sfera de cuprindere a acestui capitol. Pentru cei interesați, Internetul oferă multe tutoriale despre aceste noțiuni[1].

11.5 Automatizarea sarcinilor – make

Aplicațiile complexe dispun de un număr mare de fișiere sursă. Compilarea și link-editarea acestora poate deveni un proces anevoios și repetitiv. Se poate întâmplă ca doar câteva fișiere sursă să fie modificate și să nu fie nevoie de recompilarea întregii aplicații. Se dorește, așadar, un utilitar care să automatizeze sarcinile de compilare, linking și altele și să rezolve dependențele între fișierele sursă pentru a eficientiza procesul.

Make este un un astfel de utilitar: un program pentru automatizarea task-urilor. Make rezolvă problema execuției unor acțiuni în funcție de relațiile (dependețele) între ele. O acțiune este executată doar dacă acțiunile de care depinde au fost executate.

Relațiile între acțiuni sunt definite într-un fișier numit `Makefile`. Pe baza acestuia, Make determină automat secvența de pași care trebuie efectuată pentru respectarea dependențelor. Pentru a optimiza execuția acțiunilor, Make determină dacă o acțiune a fost efectuată și nu o mai execută. În același timp, Make detectează dacă pentru o

[1] http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html

acțiune s-a modificat o dependență și o execută din nou. Figura 11.5 descrie mdoul de funcționare a utilitarului Make.

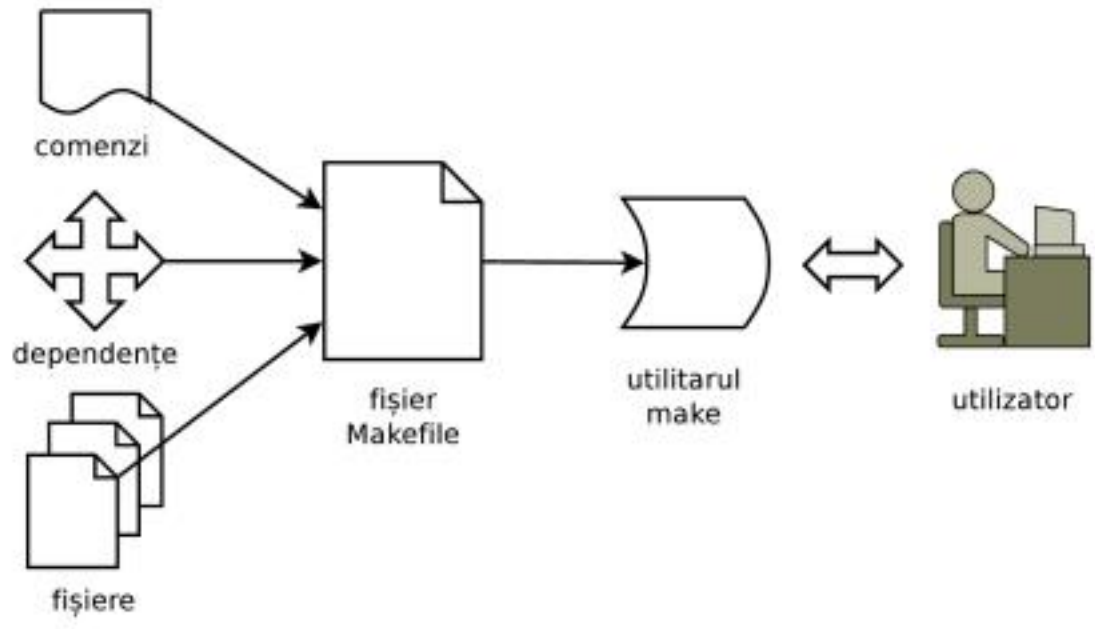


Figura 11.5: Funcționare Make

Make reprezintă, de fapt, o clasă de aplicații. Cel mai răspândit membru al acestei clase este GNU Make[1]. Executabilul asociat este `make`.

Vom descrie modul de funcționare și utilizare a Make prin compilarea și link-editarea aplicației descrisă în secțiunea 10.3. Aplicația constă din 4 fișiere (`main.c`, `sum.c`, `sumalt.c`, `func.h`).

11.5.1 Cel mai simplu Makefile

Cea mai simplă formă de fișier `Makefile` conține comanda de compilare și linking a fișierelor de mai sus:

```
1 exec:
2         gcc main.c sum.c sumalt.c -o exec
```

Listing 11.13: Makefile foarte simplu

Pentru rularea comenzii descrise în fișierul `Makefile` se folosește comanda **`make`**:

```
1 razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ make
2 gcc main.c sum.c sumalt.c -o exec
```

Fișierul Makefile specifică faptul că pentru a obține executabilul `exec` se rulează comanda **`gcc main.c sum.c sumalt.c -o exec`**. `exec` poartă numele de **target** (*țintă*) a fișierului Makefile: ceea ce se dorește obținut.

Din motive de tradiție, comenzile dintr-un fișier Makefile sunt precedate de caracterul `TAB`. O greșeală frecventă este folosirea de spații în loc de caracterul `TAB` înainte de folosirea unei comenzi. Apariția unui mesaj de forma celui de mai jos înseamnă, destul de probabil, omiterea folosirii caracterului `TAB`:

```
1 razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ make
2 Makefile:2: *** missing separator.  Stop.
```

[1] http://www.gnu.org/software/make/

11.5.2 Folosirea dependențelor

Fișierul Makefile de mai sus are un neajuns important: la orice rulare a comenzii Make se va executa comanda de compilare și linking indiferent dacă fișierele sursă au fost sau nu modificate. Acest lucru înseamnă realizarea unei acțiuni inutile.

Pentru a evita acest lucru, specificăm în Makefile faptul că executabilul `exec` se obține din cele trei fișiere sursă (`main.c`, `sum.c`, `sumalt.c`). Cele trei fișiere sursă sunt adăugate în lista de dependențe a executabilului (după caracterul `:` - *două puncte*). **Dependențele** sunt fișiere sau alte target-uri de a căror existență sau realizare depinde target-ul curent:

```
1 exec: main.c sum.c sumalt.c
2         gcc main.c sum.c sumalt.c -o exec
```

Listing 11.14: Makefile cu dependențe

În această situație, rularea comenzii **`make`** nu mai are ca efect compilarea și link-editarea celor trei fișiere sursă ci afișarea unui mesaj care menționează că executabilul `exec` există și nu au fost modificate fișierele de care depinde:

```
1 razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ make
2 make: 'exec' is up to date.
```

Dacă un fișier din lista de dependențe va fi modificat, Make va observa acest lucru și va rula din nou comanda. Un element target sau dependență a unui fișier Makefile are două stări:

- actualizat: fișierul asociat nu a fost modificat și dependențele sale sunt actualizate;
- neactualizat: fișierul asociat a fost modificat sau o parte din dependențele sale nu sunt actualizate.

11.5.3 Dependențe ierarhice

Fișierul Makefile anterior are încă neajunsuri. Un neajuns important este prezența fișierelor sursă în lista de dependențe.

Fie situația în care programatorul modifică fișierul `sum.c`. Rularea comenzii **`make`** va însemna următorul set de pași:

- se verifică dacă dependențele target-ului `exec` sunt actualizate;
- se observă că fișierul `sum.c` a fost modificat, deci dependența `sum.c` este neactualizată;
- target-ul `exec` este marcat ca neactualizat; trebuie rulată comanda asociată target-ului;
- se rulează comanda **`gcc main.c sum.c sumalt.c`**;
- comanda are drept consecință compilarea **tuturor** fișierelor din lista de dependențe și apoi link-editarea fișierelor obiect asociate.

Se efectuează, așadar, trei procese de compilare (pentru `main.c`, `sum.c`, `sumalt.c`) și un proces de link-editare a celor trei module obiect asociate. Totuși, întrucât doar fișierul `sum.c` a fost modificat, nu este nevoie de compilarea fișierelor `main.c` și `sumalt.c`.

Pentru a elimina acest neajuns, se modifică fișierul `Makefile` astfel:

```
1  exec: main.o sum.o sumalt.o
2          gcc main.o sum.o sumalt.o -o exec
3
4  main.o: main.c
5          gcc -c main.c -o main.o
6
7  sum.o: sum.c
8          gcc -c sum.c -o sum.o
9
10 sumalt.o: sumalt.c
11         gcc -c sumalt.c -o sumalt.o
```

Listing 11.15: Makefile cu dependențe ierarhice

În această situație, fișierul `Makefile` descrie o arborescență de dependențe, ca în figura 11.6.

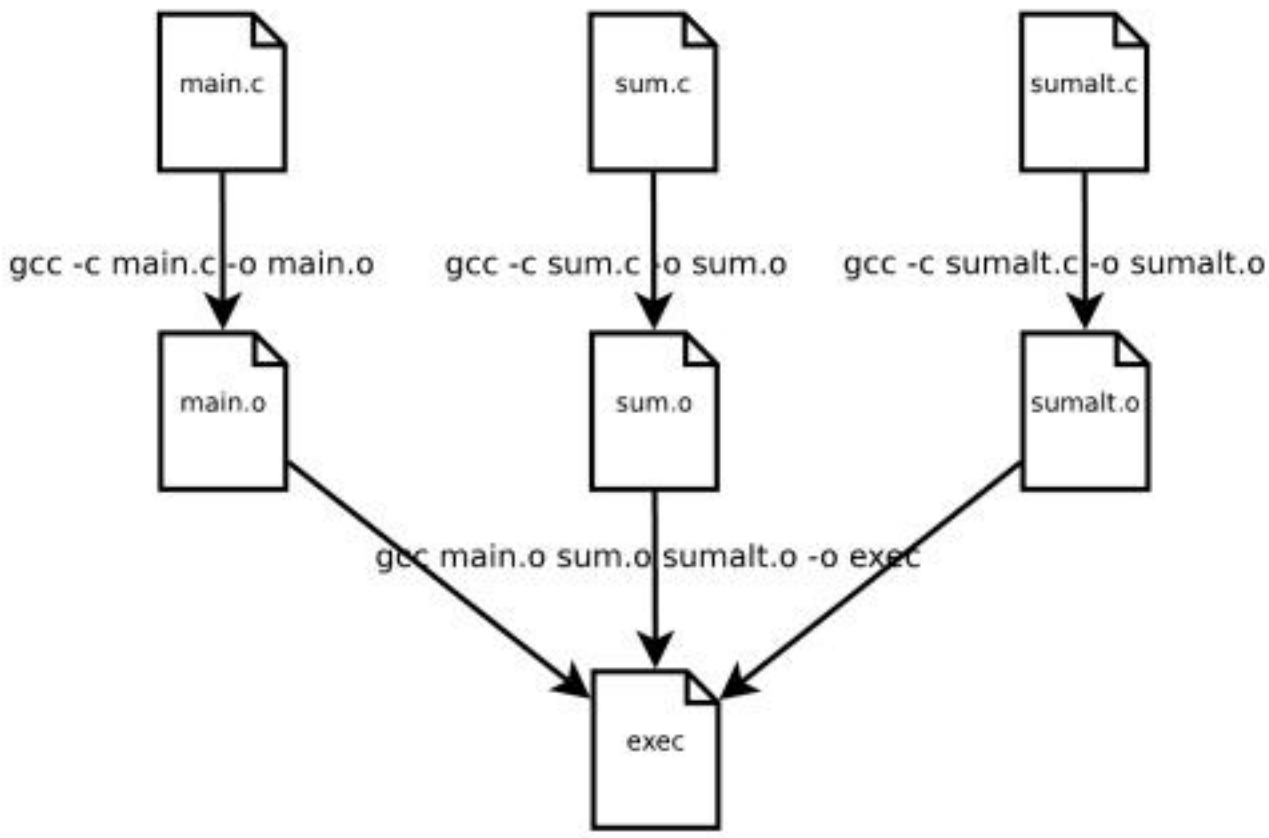


Figura 11.6: Dependențe într-un fișier Makefile

În figură sunt reprezentate și comenzile rulate pentru obținerea fiecărui fișier în parte. Astfel, o rulare simplă a comenzii **`make`** va produce următorul rezultat:

```
1  razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ make -f Makefile.
   dep.ier
2  gcc -c main.c -o main.o
3  gcc -c sum.c -o sum.o
4  gcc -c sumalt.c -o sumalt.o
5  gcc main.o sum.o sumalt.o -o exec
```

Pașii pe care îi efectuează utilitarul Make sunt:

- verifică lista de dependențe a target-ului `exec`; există trei dependențe care au la rândul lor alte dependențe; aceste dependențe trebuie verificate;
- target-ul `main.o` nu există, deci trebuie obținut; se rulează comanda **`gcc -c main.c -o main.o`**;
- target-ul `sum.o` nu există, deci trebuie obținut; se rulează comanda **`gcc -c sum.c -o sum.o`**;
- target-ul `sumalt.o` nu există, deci trebuie obținut; se rulează comanda **`gcc -c sumalt.c -o sumalt.o`**;
- toate cele trei dependențe ale target-ului `exec` erau neactuzalizate; target-ul `exec` este atunci neactualizat și trebuie rulată comanda asociată: **`gcc main.o sum.o sumalt.o -o exec`**.

După cum se precizase la început, presupunem că programatorul modifică fișierul `sum.c`. Rezultatul rulării comenzii **`make`** este:

```
1 razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ make -f Makefile.
  dep.ier
2 gcc -c sum.c -o sum.o
3 gcc main.o sum.o sumalt.o -o exec
```

Pașii urmați de utilitarul Make sunt:

- verifică lista de dependențe a target-ului `exec`; există trei dependențe care au la rândul lor alte dependențe; cele trei dependențe trebuie verificate;
- target-ul `main.o` există, dependența sa (`main.c`) există și este nemodificată; target-ul este actualizat, nu este nevoie de rularea comenzii asociate;
- target-ul `sum.o` există, dependența sa (`sum.c`) există **dar este modificată**; target-ul `sum.o` trebuie actualizat și se rulează comandă **`gcc -c sum.c -o sum.o`**;
- target-ul `sumalt.o` există, dependența sa (`sumalt.c`) există și este nemodificată; target-ul este actualizat, nu este nevoie de rularea comenzii asociate;
- una dintre trei dependențe ale target-ului `exec` (`sum.o`) este neactuzalizată; target-ul `exec` este atunci neactualizat și trebuie rulată comanda asociată: **`gcc main.o sum.o sumalt.o -o exec`**.

Se observă că, în acest caz, din cei patru pași posibili (trei pași de compilare și un pas de link-editare) este nevoie de execuția a doar doi dintre aceștia: compilarea `sum.c` și link-editarea.

Fișierul `Makefile` curent mai prezintă un neajuns: în cazul unei modificări a fișierul header `func.h` nu se actualizează niciun target, întrucât fișierul nu este adăugat în lista de dependențe a niciunui target. Un fișier `Makefile` corect arată astfel:

```
1 exec: main.o sum.o sumalt.o
2         gcc main.o sum.o sumalt.o -o exec
3
4 main.o: main.c func.h
5         gcc -c main.c -o main.o
6
```

```
7  sum.o: sum.c func.h
8          gcc -c sum.c -o sum.o
9
10 sumalt.o: sumalt.c func.h
11         gcc -c sumalt.c -o sumalt.o
```

Listing 11.16: Makefile cu fișier header în lista de dependențe

11.5.4 Target-ul clean

Pentru că, de obicei, se dorește curățarea completă a mediului de lucru (pentru distribuția surselor, mutarea aplicației, recompilarea acesteia în alt mediu etc.), un fișier Makefile oferă, de obicei, o regulă numită `clean`. Regula are rolul de a șterge fișierele produse în urma rulării comenzilor din fișierul Makefile: fișiere obiect, fișiere executabile, fișiere temporare etc.

În mod tipic, o regulă `clean` are asociată o comandă **`rm`** ca mai jos:

```
1  exec: main.o sum.o sumalt.o
2          gcc main.o sum.o sumalt.o -o exec
3
4  main.o: main.c
5          gcc -c main.c -o main.o
6
7  sum.o: sum.c
8          gcc -c sum.c -o sum.o
9
10 sumalt.o: sumalt.c
11         gcc -c sumalt.c -o sumalt.o
12
13 clean:
14         rm -f exec main.o sum.o sumalt.o
```

Listing 11.17: Makefile cu target clean

Pentru a rula comanda asociată target-ului `clean` se transmite target-ul ca argument comenzii **`make`**. Astfel, în urma rulării comenzii **`make clean`** se șterg fișierele obiect și fișierul executabil:

```
1  razvan@valhalla:~/carte-uso.git/src/code/10-compilare/tmp$ ls
2  Makefile  exec  func.h  main.c  main.o  sum.c  sum.o  sumalt.c  sumalt.o
3
4  razvan@valhalla:~/carte-uso.git/src/code/10-compilare/tmp$ make clean
5  rm -f exec main.o sum.o sumalt.o
6  razvan@valhalla:~/carte-uso.git/src/code/10-compilare/tmp$ ls
7  Makefile  func.h  main.c  sum.c  sumalt.c
```

11.5.5 Target-urile .PHONY și all

O problemă care poate împiedica funcționarea corectă a target-ului `clean` este prezența unui fișier cu numele `clean` în directorul curent. În această situația, Make va considera target-ul `clean` ca fiind actualizat și nu va rula comanda asociată.

```
1 razvan@valhalla:~/carte-uso.git/src/code/10-compilare/tmp$ ls
2 Makefile  clean  exec  func.h  main.c  main.o  sum.c  sum.o  sumalt.c
  sumalt.o
3
4 razvan@valhalla:~/carte-uso.git/src/code/10-compilare/tmp$ make clean
5 make: 'clean' is up to date
```

Pentru a rezolva această problemă, trebuie marcat `clean` ca un target care nu este niciodată actualizat. Comanda asociată va fi, astfel, executată tot timpul. Marcajul unui target ca *neactualizabil* se realizează cu ajutorul target-ului predefinit `.PHONY`, ca mai jos:

```
1 .PHONY: clean
2
3 clean:
4         rm -f exec main.o sum.o sumalt.o
```

Listing 11.18: Makefile cu target clean

Folosirea target-ului `.PHONY` înseamnă că se va rula comanda asociata target-ului `clean` indiferent dacă există sau nu un fișier cu numele `clean` în directorul curent:

```
1 razvan@valhalla:~/carte-uso.git/src/code/10-compilare/tmp$ ls
2 Makefile  clean  exec  func.h  main.c  main.o  sum.c  sum.o  sumalt.c
  sumalt.o
3
4 razvan@valhalla:~/carte-uso.git/src/code/10-compilare/tmp$ make clean
5 rm -f exec main.o sum.o sumalt.o
```

Target-ul `.PHONY` este, în general, folosit și pentru marcarea target-ului `all` ca target *neactulizabil*. Target-ul `all` este, în general, primul target într-un fișier Makefile și este verificat implicit la rularea comenzii **`make`**. Target-ul `all` încapsulează ca dependențe fișierele/target-urile ce se doresc obținute prin rularea comenzii **`make`** (în cazul de față este vorba doar de target-ul `exec`).

```
1  .PHONY: all clean
2
3  all: exec
4
5  exec: main.o sum.o sumalt.o
6          gcc main.o sum.o sumalt.o -o exec
7
8  main.o: main.c
9          gcc -c main.c -o main.o
10
11 sum.o: sum.c
12         gcc -c sum.c -o sum.o
13
14 sumalt.o: sumalt.c
15         gcc -c sumalt.c -o sumalt.o
16
17 clean:
18         rm -f exec main.o sum.o sumalt.o
```

Listing 11.19: Makefile cu target all

11.5.6 Variabile în Makefile

În fișierele Makefile construite până acum nu a fost folosită opțiunea `-Wall` la compilarea fișierelor sursă C. O soluție simplă este introducerea opțiunea `-Wall` în cadrul celor trei comenzi de compilare. Totuși, această abordare nu este flexibilă. Prespunând că, la un moment ulterior, se dorește și adăugare opțiunea `-g` pentru depanare, vor trebui din nou actualizate cele trei comenzi.

O abordare flexibilă este folosirea variabilelor în cadrul fișierului Makefile. O variabilă se definește o dată și se folosește de mai multe ori. Dacă, la un moment ulterior, se dorește actualizarea variabilei, modificarea se va face doar în locul unde a fost definită.

```
1  CC = gcc
2  CFLAGS = -Wall
3
4  exec: main.o sum.o sumalt.o
5          $(CC) main.o sum.o sumalt.o -o exec
6
7  main.o: main.c
8          $(CC) $(CFLAGS) -c main.c -o main.o
9
10 sum.o: sum.c
11         $(CC) $(CFLAGS) -c sum.c -o sum.o
12
13 sumalt.o: sumalt.c
14         $(CC) $(CFLAGS) -c sumalt.c -o sumalt.o
15
16 clean:
17         rm -f exec main.o sum.o sumalt.o
```

Listing 11.20: Makefile cu variabile

În cadrul fișierului Makefile de mai sus, au fost folosite variabilele, respectiv, `CC` pentru a reține comanda asociată compilatorului și `CFLAGS` pentru a reține opțiunile transmise acestuia. Variabila `CFLAGS` nu a fost folosită pentru obținerea executabilului `exec` pentru că are sens doar în faza de compilare, nu și în cea de link-editare (avertismentele indicate de `-Wall` rezultă în urma etapei de compilare).

11.5.7 Sintaxă Makefile

Această secțiune are un rol recapitulativ și descrie formal sintaxa unui fișier Makefile. După cum s-a prezentat și în secțiunile anterioare, sintaxa unui fișier Makefile este compusă dintr-o listă de reguli de forma:

```
1  target: dependente ...
2          <tab>comanda
3          <tab>comanda
4          <tab>...
```

Listing 11.21: Sintaxă Makefile

unde:

- **target** – numele unui fișier care trebuie generat sau numele unei acțiuni; în cazul în care target-ul este un fișier, comenzile trebuie să genereze acest fișier; în cazul

în care comenzile nu generează un fișier cu numele target-ului, se va considera la fiecare rulare ca target-ul nu a fost creat (așadar toate comenzile se vor executa);

- **dependențe** – lista de target-uri (separate prin spații) care trebuie îndeplinite (fișiere care trebuie să existe) pentru a se realiza target-ul curent – make poate determina dacă trebuie reexecutat target-ul dacă una dintre dependețe s-a modificat (unul din fișierele din lista de dependențe s-a modificat);
- **comandă** – comanda care duce la realizarea target-ului (de obicei la comenzi sunt trecute comenzile de compilare care duc la realizarea unui fișier cu numele target-ului);
- **<tab>** – caracterul `TAB`.

Regulile formează un arbore de dependențe. Pentru ca un target să fie executat, se verifică dacă toate dependențele lui există ca fișiere. Dacă nu este îndeplinită această condiție, dependențele care nu există sunt executate, aplicându-se același algoritm.

11.5.8 Moduri de utilizare a Make

La execuția comenzii **`make`**, vor fi luate în considerare target-urile definite în fișierul `Makefile`.

Pentru a executa un target, se folosește sintaxa:

```
1 mircea@cougar:~/carte-uso/cap-10$ make target
```

unde `target` este numele target-ului care se dorește a fi executat.

La utilizarea lui Make fără parametri, se va executa primul target definit în fișierul `Makefile`. Din acest motiv, primul target descris este de obicei cel mai complex, cu cele mai multe dependențe. Convențional, Makefile-urile folosite pentru compilarea unei aplicații definesc primul target `all`. Target-ul `all` deține în lista de dependențe celelalte target-uri care trebuie obținute.

Utilitarul GNU Make folosește, în mod implicit, unul din fișierele `GNUMakefile`, `Makefile` sau `makefile`. În cazul în care niciunul dintre acestea nu există, și comanda **`make`** nu primește nici un argument, va întoarce eroare:

```
1 razvan@valhalla:~/carte-uso.git/src/code$ make
2 make: *** No targets specified and no makefile found.  Stop.
```

Pentru a forța folosirea unui fișier Makefile cu alt nume decât numele implicite, se folosește opțiunea `-f`:

```
1 razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ make -f Makefile.
  dep.ier.head
2 gcc -c main.c -o main.o
3 gcc -c sum.c -o sum.o
4 gcc -c sumalt.c -o sumalt.o
5 gcc main.o sum.o sumalt.o -o exec
```

Make poate funcționa și fără existența unui fișier Makefile. Pentru a compila direct fișierul `myprog.c` (care nu depinde de altele) se folosește comanda:

```
1 mircea@cougar:~/carte-uso/cap-10$ make myprog
```

Iar pentru a compila direct `myprog.c` cu parametrul de compilare `-Wall` se folosește comanda:

```
1 mircea@cougar:~/carte-uso/cap-10$ make CFLAGS=-Wall myprog
```

11.6 Portabilitate

Portabilitatea este caracteristica unei aplicații de a putea fi folosită într-un **mediu diferit** de cel pentru care fost inițial proiectată. Acest lucru poate însemna un alt sistem de operare, o altă arhitectură hardware, o altă bibliotecă.

Se spune că o aplicație este portabilă dacă are nevoie de modificări minime pentru a putea rula pe un alt mediu. Acțiunea de portare este acțiunea de modificare a unei aplicații pentru a putea fi folosită pe un alt sistem de operare sau sistem fizic. Un mediu pe care se realizează portarea se mai numește platformă.

Dacă în deceniile trecute numărul de arhitecturi posibile pe care rula o aplicație era destul de mare, în zilele noastre arhitectura hardware dominantă este x86. La fel sistemele de operare pe care se portează o aplicație sunt Microsoft Windows, Apple Mac OS X și Unix/Linux. În lumea sistemelor embedded, însă, portabilitatea rămâne o problemă importantă.

11.6.1 Portabilitatea la nivelul arhitecturii sistemului de calcul

În zilele noastre, sistemele desktop sunt dominate de arhitectura x86 așa că rareori este nevoie de portarea unei aplicații la nivelul arhitecturii sistemului de calcul. În sistemele embedded portarea la nivelul procesorului este un factor important.

În general, însă, portarea la nivelul arhitecturii nu este necesară datorită compilatorului. Astfel, pentru un limbaj de nivel înalt, compilatorul va asigura traducerea codului sursă în cod mașină specific procesorului dat. GCC este cel mai portat și cel mai portabil compilator fiind capabil de a compila programe pentru o varietate de arhitecturi.

11.6.2 Portabilitatea unui limbaj de programare

Programarea în limbaj de asamblare nu asigură, prin definiție, portabilitatea codului, deorece acesta este intrinsec legat de arhitectura sistemului de calcul. Unul dintre motivele răspândirii Unix în anii '70 a fost scrierea acestuia în C, un limbaj de nivel înalt portabil.

Folosirea C însemna eliberarea de arhitectura hardware pe care va rula aplicația. Portarea codului era făcută prin intermediul compilatorului. În ziua de azi, aplicațiile C sunt ușor portabile datorită existenței unui compilator C pe orice platformă. Cu toate acestea, anumite aspecte precum organizarea octeților[1] sau lungimea cuvântului

[1] http://en.wikipedia.org/wiki/Endianness

procesorului[1] rămân aspecte de care trebuie ținut cont.

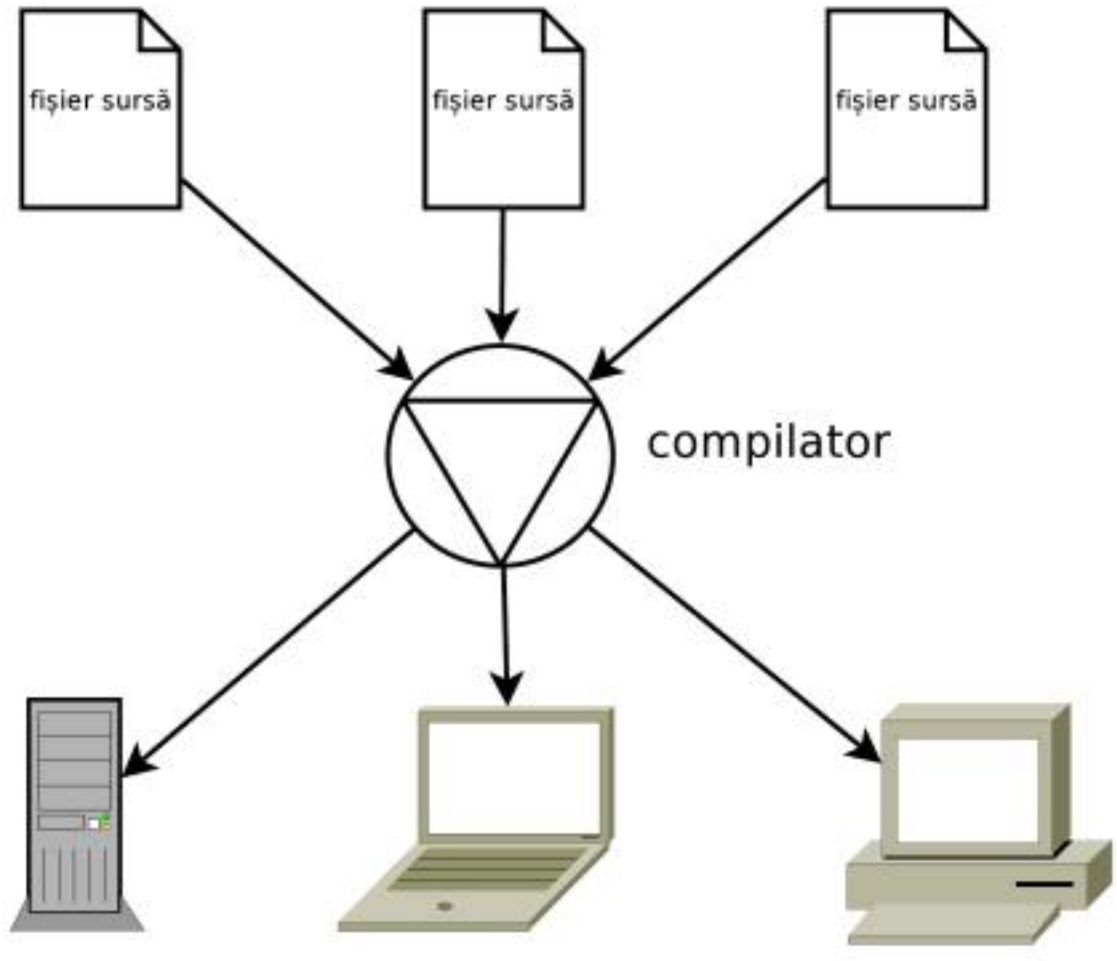


Figura 11.7: Portabilitatea asigurată de compilator

Java este un limbaj de programare gândit pentru a fi portabil. Deviza Java este "Write once, run everywhere". Pentru a asigura acest lucru, un program Java este compilat într-o formă portabilă de modul obiect denumită bytecode. Acest bytecode este apoi interpretat de o mașină virtuală Java. Mașina virtuală Java este responsabilă cu translatarea bytecode-ului portabil în cod specific sistemului pe care acesta rulează. Astfel, rularea unui program Java sau unui modul compilat în bytecode pe o nouă platformă este condiționată de existența unei mașini virtuale pe acea platformă. Mașina virtuală Java este conținută în pachetul JRE de la Sun (Java Runtime Environment). Acesta conține mașina virtuală și bibliotecile cu funcții standard Java.

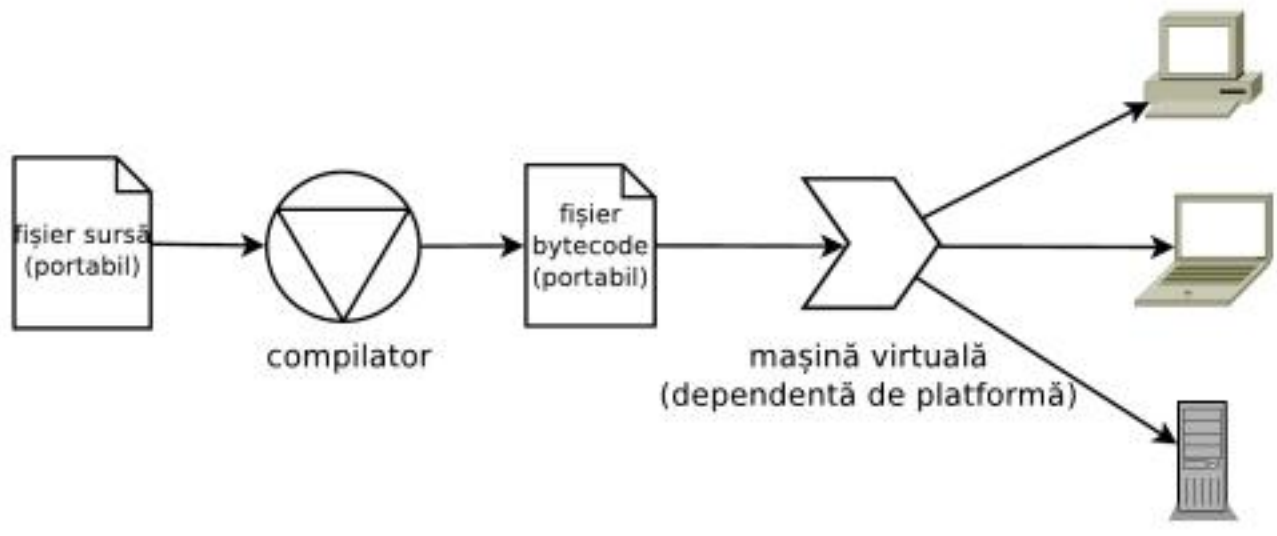


Figura 11.8: Portabilitatea asigurată de mașina virtuală

[1] http://en.wikipedia.org/wiki/Word_(computing)

11.6.3 Portabilitatea la nivelul sistemului de operare

Portabilitatea unei aplicații se poate referi și la posibilitatea rulării acesteia pe un alt sistem de operare. Acest lucru este corelat, de obicei, cu interfețele de programare (API – Application Programming Interface) pe care sistemul de operare le pune la dispoziție (system API). Astfel, Windows pune la dispoziția programatorului C interfața Win32 API[1]. Unix, pe de altă parte pune la dispoziția programatorului interfața POSIX (Portable Operating System Interface)[2]. Pentru deschiderea unui fișier un programator Windows va folosi apelul `CreateFile`, iar un programator Unix `open`.

Pentru asigurarea portabilității însă, multe biblioteci oferă programatorului o interfață portabilă peste sistemul de operare. Astfel, funcțiile ANSI din biblioteca standard C sunt portabile peste diverse sisteme de operare. De exemplu, pentru deschiderea unui fișier, programatorul va folosi apelul `fopen` indiferent de sistemul de operare pe care va rula aplicația.

Un exemplu de bibliotecă portabilă este wxWidgets[3]. Această bibliotecă permite crearea de aplicații grafice peste diferite sisteme de operare prin intermediul unui API comun. Biblioteca ascunde atât comunicarea cu sistemul de operare din spate cât și cu bibliotecile grafice neportabile pe care acesta le pune în mod obișnuit la dispoziție.

11.7 Studiu de caz

11.7.1 GCC în Windows

Suportul pentru compilatorul GCC în mediul Windows este disponibil prin proiectele:

- MinGW[4] (Minimalist GNU for Windows) reprezintă o portare a aplicațiilor de dezvoltare de bază ale proiectului GNU pe sisteme Windows. Componentele principale sunt GNU Compiler Collection (GCC), GNU Binutils și bibliotecile de runtime C. MinGW folosește DLL-urile de sistem standard și API-ul de bază al sistemului Windows.
- Cygwin[5] este un mediu care oferă aplicațiilor un strat (layer) de emulare a API-ului Unix, împreună cu o colecție de utilitare care oferă funcționalitățile din Unix. În vreme ce MinGW oferă doar aplicații pentru dezvoltare, Cygwin permite instalarea unei mari diversități de aplicații.

Aceste implementări oferă acces la suita de aplicații de dezvoltare a proiectului GNU. Parametrii și modul de utilizare a compilatorului sunt similare unui sistem Linux.

În mediul Windows este posibilă utilizarea GCC și împreună cu un mediu integrat de dezvoltare – IDE. Dev-C++[6] este un astfel de IDE care oferă posibilitatea compilării programelor utilizând GCC.

---

[1] http://msdn.microsoft.com/en-us/library/aa383749(VS.85).aspx
[2] http://www.opengroup.org/onlinepubs/9699919799/
[3] http://www.wxwidgets.org/
[4] http://www.mingw.org/
[5] http://sourceware.org/cygwin/
[6] http://www.bloodshed.net/devcpp.html

11.7.2 Link-editarea modulelor C și a modulelor C++

Deși două limbaje asemănătoare (C++ este considerat un superset al C), combinarea codului scris în C și C++ poate furniza probleme. În cele ce urmează nu ne vom referi la accesarea claselor C++ din C sau folosirea apelurilor din biblioteca standard C++ în programe C – situații rar întâlnite. Ne vom referi la folosirea funcțiilor definite în module C într-un modul C++ și invers[12].

Contextul

Vom considera un scenariu foarte simplu care include 6 fișiere:

- `c_print.c`: fișier C conține implementarea funcției `c_print`;
- `cpp_print.cpp`: fișier C++ conține implementarea funcției `cpp_print`;
- `c_print.h`: fișier header ce conține declarația funcției `c_print`;
- `cpp_print.h`: fișier header ce conține declarația funcției `cpp_print`;
- `c_main.c`: fișier C conținând funcția `main`; va apela funcția `c_print` din fișierul `c_print.c` sau funcția `cpp_print` din fișierul `cpp_print.cpp`;
- `cpp_main.cpp`: fișier C++ conținând funcția `main`; va apela funcția `c_print` din fișierul `c_print.c` sau funcția `cpp_print` din fișierul `cpp_print.cpp`.

```c
#include <stdio.h>

#include "c_print.h"

void c_print(void)
{
        printf("C-style: Wassup!\n");
}
```

Listing 11.22: c_print.c

```cpp
#include <iostream>

#include "cpp_print.h"

void cpp_print()
{
        std::cout << "C++ style: True, true!" << std::endl;
}
```

Listing 11.23: cpp_print.cpp

```c
#ifndef C_PRINT_H_
#define C_PRINT_H_      1

void c_print(void);

#endif
```

[1] http://en.wikipedia.org/wiki/Compatibility_of_C_and_C%2B%2B#Linking_C_and_C.2B.2B_code
[2] http://www.parashift.com/c++-faq-lite/mixing-c-and-cpp.html

Listing 11.24: c_print.h

```
#ifndef CPP_PRINT_H_
#define CPP_PRINT_H_    1

void cpp_print();

#endif
```

Listing 11.25: cpp_print.h

```
#include "c_print.h"
//#include "cpp_print.h"

int main(void)
{
        c_print();
        //cpp_print();

        return 0;
}
```

Listing 11.26: c_main.c

```
#include "cpp_print.h"
//#include "c_print.h"

int main()
{
        cpp_print();
        //c_print();

        return 0;
}
```

Listing 11.27: cpp_main.cpp

În mod implicit, în cadrul fișierului `c_main.c` se apelează funcția `c_print`, iar în cadrul fișierului `cpp_main.cpp` se apelează funcția `cpp_main`.

Într-un prim pas, se compilează cele patru fișiere C/C++. Folosim **gcc** pentru compilarea fișierelor C și **g++** pentru compilarea fișierelor C++.

```
razvan@valhalla:~/mix$ ls
c_main.c  c_print.c  c_print.h  cpp_main.cpp  cpp_print.cpp  cpp_print.h

razvan@valhalla:~/mix$ gcc -Wall -c c_main.c

razvan@valhalla:~/mix$ gcc -Wall -c c_print.c

razvan@valhalla:~/mix$ g++ -Wall -c cpp_main.cpp

razvan@valhalla:~/mix$ g++ -Wall -c cpp_print.cpp

razvan@valhalla:~/mix$ ls
c_main.c  c_print.c  c_print.o     cpp_main.o     cpp_print.h
c_main.o  c_print.h  cpp_main.cpp  cpp_print.cpp  cpp_print.o
```

Pentru link-editare, folosim perechile `c_main.o` și `c_print.o`, respectiv `cpp_main.o` și `cpp_print.o`. Executabilele obținute sunt denumite în conformitate cu fișierele obiect folosite pentru generare.

```
razvan@valhalla:~/mix$ gcc c_main.o c_print.o -o c_main-c_print

razvan@valhalla:~/mix$ g++ cpp_main.o cpp_print.o -o cpp_main-cpp_print

razvan@valhalla:~/mix$ ./c_main-c_print
C-style: Whassup?

razvan@valhalla:~/mix$ ./cpp_main-cpp_print
C++ style: True, true!
```

Pentru link-editarea modulelor obiect obținute din surse C++ a fost folosită comanda **g++**. Comanda **g++** apelează linker-ul (**ld**) cu toate opțiunile necesare. Se poate folosi și comanda **gcc** pentru linking, dar trebuie specificată explicit folosirea bibliotecii standard C++ (`libstdc++`).

```
razvan@valhalla:~/mix$ gcc cpp_main.o cpp_print.o -o cpp_main-cpp_print
cpp_main.o:(.eh_frame+0x12): undefined reference to `__gxx_personality_v0
'
cpp_print.o: In function `cpp_print()':
cpp_print.cpp:(.text+0xa): undefined reference to `std::cout'
[...]
collect2: ld returned 1 exit status

razvan@valhalla:~/mix$ gcc cpp_main.o
cpp_print.o -o cpp_main-cpp_print_new -lstdc++

razvan@valhalla:~/mix$ ./cpp_main-cpp_print_new
C++ style: True, true!
```

Problema

Ce se întâmplă, însă, dacă link-editarea se realizează între un modul obiect provenit dintr-un fișier sursă C și un modul provenit dintr-un fișier sursă C++?

Pentru a testa acest lucru, sunt actualizate fișierele `c_main.c`, respectiv `cpp_main.cpp` pentru a include și apela funcțiile specifice celuilalt limbaj. După care se compilează și se realizează link-editarea:

```
razvan@valhalla:~/mix-new$ gcc c_main.o cpp_print.o -o c_main-cpp_print -
lstdc++
c_main.o: In function `main':
c_main.c:(.text+0xa): undefined reference to `cpp_print'
collect2: ld returned 1 exit status

razvan@valhalla:~/mix-new$ g++ c_main.o cpp_print.o -o c_main-cpp_print -
lstdc++
c_main.o: In function `main':
c_main.c:(.text+0xa): undefined reference to `cpp_print'
collect2: ld returned 1 exit status

razvan@valhalla:~/mix-new$ g++ cpp_main.o c_print.o -o cpp_main-c_print
cpp_main.o: In function `main':
cpp_main.cpp:(.text+0x5): undefined reference to `c_print()'
```

```
14 collect2: ld returned 1 exit status
15
16 razvan@valhalla:~/mix-new$ gcc cpp_main.o c_print.o -o cpp_main-c_print -
   lstdc++
17 cpp_main.o: In function 'main':
18 cpp_main.cpp:(.text+0x5): undefined reference to 'c_print()'
19 collect2: ld returned 1 exit status
```

În ambele situații, indiferent de comanda folosită (**gcc** sau **g++**) se obține eroare de linker. Linker-ul nu găsește simbolul asociat funcției `cpp_print`, respectiv `c_print`.

Folosind utilitarul **nm** se pot inspecta simbolurile din cadrul celor patru module obiect:

```
1  razvan@valhalla:~/mix-new$ nm c_main.o
2                  U cpp_print
3  0000000000000000 T main
4
5  razvan@valhalla:~/mix-new$ nm cpp_main.o
6                  U _Z7c_printv
7                  U __gxx_personality_v0
8  0000000000000000 T main
9
10 razvan@valhalla:~/mix-new$ nm c_print.o
11 0000000000000000 T c_print
12                 U puts
13
14 razvan@valhalla:~/mix-new$ nm cpp_print.o
15 000000000000005f t _GLOBAL__I__Z9cpp_printv
16 0000000000000022 t _Z41__static_initialization_and_destruction_0ii
17 0000000000000000 T _Z9cpp_printv
18 [...]
```

Problema este folosirea tehnicii de *name mangling*[1] folosită de compilatorul de C++. Simbolurile dintr-un modul C++ sunt precedate de o construcție specială de formă `_Z`. Tehnica este folosită pentru a preveni folosirea aceluiași identificator pentru o variabilă funcție care are același nume în două namespace-uri diferite.

Modulul `c_main.o` marchează simbolul `cpp_print` nedefinit și solicită linker-ului rezolvarea acestuia. Modulul `cpp_print.o` definește însă simbolul `_Z9cpp_printv` și linker-ul nu poate face rezolvarea. Similar se întâmplă și în cazul simbolului `c_print` (varianta C) și `_Z7c_printv` (varianta C++).

Soluția

Pentru a rezolva această problemă, compilatorul trebuie să dețină informații suplimentare și să marcheze simbolurile corespunzătoare în modulele obiect.

O soluție directă este compilarea tuturor fișierelor sursă folosind **g++**. Compilatorul de C++ poate compila și fișiere C, iar modulele obținute vor folosi tehnica de *name mangling*.

```
1 razvan@valhalla:~/mix-new$ g++ cpp_main.cpp c_print.c -o cpp_main-c_print
2
3 razvan@valhalla:~/mix-new$ ./cpp_main-c_print
```

[1] http://en.wikipedia.org/wiki/Name_mangling

```
C-style: Whassup?

razvan@valhalla:~/mix-new$ g++ c_main.c cpp_print.cpp -o c_main-cpp_print

razvan@valhalla:~/mix-new$ ./c_main-cpp_print
C++ style: True, true!
```

Întrucât este posibilă absența unui anumit tip de compilator de pe un sistem și pentru că este mai eficient ca un fișier sursă într-un limbaj de prorgramare să fie compilat cu un compilator specific, este de dorit o altă soluție. Soluția constă în folosirea construcției `extern "C"`. Această construcție se plasează înaintea declarației unei funcții pentru a forța compilatorul de C++ să folosească un identificator de simbol care să nu utilizeze tehnica de *name mangling*.

În general, construcția `extern "C"` se folosește în cadrul fișierelor header unde se găsesc declarațiile de funcții. Fisierele `c_print.h` și `cpp_print.h` actualizate vor fi:

```
#ifndef C_PRINT_H_
#define C_PRINT_H_      1

#ifdef __cplusplus
extern "C" {
#endif

void c_print(void);

#ifdef __cplusplus
}
#endif

#endif
```

Listing 11.28: c_print.h cu extern "C"

```
#ifndef CPP_PRINT_H_
#define CPP_PRINT_H_    1

#ifdef __cplusplus
extern "C" {
#endif

void cpp_print();

#ifdef __cplusplus
}
#endif

#endif
```

Listing 11.29: cpp_print.h cu extern "C"

În acest moment se poate realiza link-editarea în "diagonală", adică folosind perechea `c_main.o` și `cpp_print.o` pe de o parte și perechea `cpp_main.o` și `c_print.o` de cealaltă parte.

```
razvan@valhalla:~/mix-new$ gcc -c c_main.c

razvan@valhalla:~/mix-new$ g++ -c cpp_main.cpp
```

```
4
5  razvan@valhalla:~/mix-new$ gcc -c c_print.c
6
7  razvan@valhalla:~/mix-new$ g++ -c cpp_print.cpp
8
9  razvan@valhalla:~/mix-new$ gcc c_main.o cpp_print.o -o c_main-cpp_print -
   lstdc++
10
11 razvan@valhalla:~/mix-new$ g++ cpp_main.o c_print.o -o cpp_main-c_print
12
13 razvan@valhalla:~/mix-new$ ./c_main-cpp_print
14 C++ style: True, true!
15
16 razvan@valhalla:~/mix-new$ ./cpp_main-c_print
17 C-style: Whassup?
```

Prin folosirea utilitarului **nm** se observă folosirea unui identificator de simbol fără *name mangling* în cadrul fișierelor obiect C++.

```
1  razvan@valhalla:~/mix-new$ nm cpp_main.o
2                   U __gxx_personality_v0
3                   U c_print
4  0000000000000000 T main
5
6  razvan@valhalla:~/mix-new$ nm cpp_print.o
7  000000000000005f t _GLOBAL__I_cpp_print
8  [...]
9  0000000000000000 T cpp_print
```

Construcția `extern "C"` se folosește atât în cazul în care se apelează funcții C dintr-un modul C++ cât și în cazul apelării unei funcții C++ dintr-un modul C.

În primul caz, cel al apelării funcției C dintr-un modul C++, compilatorul de C++ marchează în modulul obiect obținut din fișierul C++ simbolul ca fiind nedefinit, în forma non-mangled. În cel de-al doilea caz, al apelării unei funcții C++ dintr-un modul C, compilatorul de C++ marchează în modulul obiect obținut din fișierul C++ simbolul ca fiind definit, în formatul non-mangled.

Compilatorul ia această decizie pe baza folosirii construcției `extern "C"`. Este important ca atât fișierul care definește funcția, cât și cel care apelează funcția să includă fișierul header care conține declarația funcției precedată de construcția `extern "C"`.

Folosirea macro-ului `__cplusplus` permite diferențierea între compilatorul de C și compilatorul de C++. Doar compilatorul de C++ va folosi construcția `extern "C"`. Construcția nu are nicio relevanță pentru compilatorul de C.

Cuvinte cheie

- compilare
- executabil
- interpretare
- limbaj
- proces
- sursă
- preprocesare
- asamblare

- cod obiect
- linker
- bibliotecă
- partajare
- portabilitate
- execuție
- apel de sistem
- make
- target
- dependență

Întrebări

1. Bibliotecile partajate de funcții:
   - ❑ sunt încărcate explicit la cerere de aplicații
   - ❑ ajută ca aplicațiile să fie mai simplu de întreținut
   - ❑ măresc aplicațiile deoarece sunt incluse în executabilele rezultate
   - ❑ sunt prezente sub formă de arhive de fișiere obiect
2. Pentru a dezactiva link-editarea implicită pe care o realizează `GCC`, se utilizează parametrul:
   - ❑ `-O0`
   - ❑ `-c`
   - ❑ `-lstdc++`
   - ❑ `-o`
3. Caracteristica unei aplicații de a putea fi folosită într-un mediu pentru care nu a fost inițial proiectată:
   - ❑ se întâlnește doar la bibliotecile partajate
   - ❑ este întâlnită numai la aplicațiile Java
   - ❑ se numește portabilitate
   - ❑ niciuna din variantele de mai sus
4. Pentru a executa un target dintr-un fișier `Makefile`, se folosește comanda
   - ❑ **`make Makefile <target>`**
   - ❑ **`make <target>`**
   - ❑ **`<target>`**
   - ❑ **`make <target> Makefile`**
5. În urma preprocesării se obțin:
   - ❑ fișiere obiect
   - ❑ fișiere sursă C fără directive de precompilare
   - ❑ fișiere în limbaj de asamblare
   - ❑ fișiere executabile

6. Care din următoarele reprezintă o bibliotecă al cărei cod este integrat explicit în codul unui executabil?
   - ❑ static library
   - ❑ shared library
   - ❑ public library
   - ❑ common library
7. Apelul de sistem reprezintă o metodă de a accesa resursele computerului. Apelurile de sistem sunt realizate doar de biblioteci.
   - ❑ adevărat, adevărat
   - ❑ adevărat, fals
   - ❑ fals, adevărat
   - ❑ fals, fals
8. Fie comanda:

```
1 mircea@cougar:~/carte-uso/cap-10$ file /lib/libm-2.3.3.so
2 /lib/libm-2.3.3.so: ELF 32-bit LSB shared object, Intel 80386,
  version 1 (SYSV), not stripped
```

   Care afirmație referitoare la fișierul `/lib/libm-2.3.3.so` este falsă?
   - ❑ fișierul este binar, nu poate fi editat cu un editor text
   - ❑ fișierul este o bibliotecă partajată
   - ❑ fișierul este executabil
   - ❑ fișierul funcționează pe o platformă Intel pe 32 biți
9. Programele interpretate se depanează mai greu decât cele compilate. În urma compilării rezultă fișiere în limbaj de asamblare care sunt executate direct pe procesor.
   - ❑ adevărat, adevarat
   - ❑ adevărat, fals
   - ❑ fals, adevărat
   - ❑ fals, fals
10. Apelurile de sistem ale programului `<program>` pot fi urmărite folosind comanda:
    - ❑ **`time <program>`**
    - ❑ **`file <program>`**
    - ❑ **`ldd <program>`**
    - ❑ **`strace <program>`**

Capitolul 12

Shell scripting

Shell to DOS... Come in DOS, do you copy? Shell to DOS...

Ce se învață din acest capitol?

- Rolul unui shell în sistem
- Ce este un script shell
- Utilitatea scripturilor shell
- Interacțiunea cu shell-ul; execuția comenzilor într-un shell
- Crearea și rularea unui script shell
- Programarea shell: variabile, funcții, structuri de control
- Folosirea filtrelor de text
- Expandarea shell
- Programarea shell (batch programming) pe Windows
- Utilitarul sed
- Utilitarul awk

12.1 Noțiuni introductive

Capitolul de față își propune prezentarea mecanismelor prin care diversele comenzi utilizate până acum pot fi combinate pentru obținerea unor efecte complexe în cadrul unui script shell.

> Un **script shell** este un program scris într-un fișier text format din combinații de comenzi și instrucțiuni specifice unui interepretor de comenzi (unui shell).

Un **shell** este un program care asigură utilizatorului o interfață de control și utilizare a sistemului de operare.

Un shell poate avea interfață grafică (GUI) sau poate avea interfață text (CLI). În cea de-a doua situație un shell mai poartă denumire de interpretor de comenzi. Shell-uri cu interfață grafică sunt *Windows Explorer* sau medii desktop construite peste *X Window System*. Shell-uri cu interfață text (interpretoare de comenzi) sunt **`cmd.exe`** (Windows NT, XP), Windows PowerShell (Windows Vista, Server 2008, 7), *Bash* (Bourne Again Shell), *csh* (C Shell).

Noțiunea de shell scripting se aplică shell-urilor cu interfață text (*Bash*, *C shell* etc.). În capitolul de față, shell-ul folosit va fi *Bash*. *GNU Bash*[1] (Bourne-Again Shell) este shell-ul implicit pe majoritatea distribuțiilor de Linux și pe Mac OS X. Ne vom referi prin *Bash* (cu majusculă) la shell, iar cu **`bash`** la executabilul utilizat de acesta.

12.1.1 De ce shell scripting?

Utilitatea principală a unui shell script derivă din faptul că este o combinație de comenzi deja existente. Acest lucru înseamnă posibilitatea de automatizare. Sarcinile repetitive pot fi descrise într-un script shell. Ori de câte ori este necesară executarea acelei sarcini, se rulează script-ul shell și aceasta va fi îndeplinită.

Un exemplu elocvent este crearea unui cont de utilizator. Pașii urmați pentru aceasta pot fi următorii:

1. crearea unui cont de utilizator (folosind **`useradd`**);
2. copierea unor directoare și fișiere pentru contul proaspăt creat;
3. stabilirea de cote (*quota*);
4. schimbarea/stabilirea parolei;
5. transmiterea unui e-mail pentru a notifica utilizatorul de crearea contului.

În lipsa unui script shell, administratorul sistemului ar trebui să ruleze comenzi pentru fiecare din pașii de mai sus ori de câte ori este necesară crearea unui cont de utilizator. Folosind un script shell, administratorul îl va rula doar pe acesta (folosind eventuali parametri). Efectul imediat este mărirea eficienței.

Scripturile shell se folosesc în general pentru automatizarea sarcinilor de rutină, precum: realizarea de backup-uri, verificarea stării sistemului etc.

Există situații când un utilizator dorește realizarea unei noi comenzi prin îmbinarea unora deja existente. Pentru a realiza acest lucru, el va crea un script shell pe care-l va putea utiliza ulterior.

În sistemele Unix, scripturile shell au un grad mare de utilizabilitate pentru că cea mai mare partea a comenzilor urmează filozofia Unix: "*Do one thing, do one thing well*". Acest lucru însemnă că fiecare comandă este utilă și (am putea spune) excelează în

[1] http://www.gnu.org/software/bash/

realizarea unei anumite sarcini, urmând ca sarcinile mai complexe să fie realizate prin "îmbinarea" mai multor comenzi simple într-un script shell.

12.1.2 Facilități oferite de scripturile shell

În afara folosirii comenzilor simple deja existente în sistem, lucru care conduce la posibilitatea de automatizare a sarcinilor, un script shell oferă și alte facilități utilizatorului.

În primul rând, un shell oferă un limbaj de programare cu variabile și instrucțiuni de ciclu (`for`, `while`) și de decizie (`if`, `case`). Acest lucru înseamnă că, dincolo de comenzile de bază, un script shell poate folosi facilitățile tipice unui limbaj de programare.

Evident, se poate pune întrebarea: De ce nu s-ar folosi un limbaj de programare precum C/Java?

Avantajul major unui script shell este timpul redus de scriere a acestuia, utilizarea unor componente deja create (fără a fi nevoie de rescrierea acestora) și folosirea unor resurse specifice ale sistemului: fișiere de configurare, drepturi de acces, utilizatori etc. Folosind comenzi deja existente, facilități de combinare a acestora și de control al fluxului, un script shell este o soluție rapidă și eficientă a unei probleme. Se spune că un script shell este o metodă "*quick and dirty*" de rezolvare a unor probleme care ar necesita efort mai mare dacă s-ar folosi alte limbaje. În plus depanarea unui script shell este extrem de facilă. În momentul în care un script shell nu funcționează corect se poate determina și corecta foarte rapid linia din script care a cauzat eroarea.

Dezavantajul unui script shell este viteza scăzută de execuție și lipsa accesului la structuri eficiente de calcul. Astfel, nu este recomandată folosirea unui script shell într-o situație în care viteza de execuție este un factor important. La fel, anumite probleme nu se pretează la a fi rezolvate cu un script shell din cauza lipsei de facilități. Problemele de algoritmică, efectuarea de calcule complexe, interacțiunea cu hardware-ul se realizează prin folosirea de limbaje specializate.

Totodată, un script shell permite interacțiunea între comenzi (sau mai bine zis între procesele create prin rularea unei comenzi), prin operatori specifici: | (pipe), ;, ||, && etc.

Alte facilități includ:

- posibilitatea reținerii ieșirii unei comenzi într-o variabilă pentru folosirea ulterioară a acesteia;
- posibilitatea redirectării intrării/ieșirii unei comenzi dintr-un/într-un fișier; acest lucru duce la prelucrarea foarte ușoară a fișierelor relevante în sistem (de exemplu: `/etc/passwd`, `/etc/shadow`);
- folosirea de comenzi non-interactive și programarea scriptului pentru execuție la un moment ulterior.

Se observă, așadar, că un script shell este un instrument foarte puternic pentru rezolvarea rapidă a diverselor probleme des întâlnite de utilizator. Programarea shell devine un factor important pentru o mai buna utilizare și gestiune a sistemului.

12.2 Interacțiunea cu shell-ul

Shell-urile cu interfață text interacționează cu utilizatorul prin intermediul comenzilor. Un shell oferă un prompt utilizatorului (în mod tipic acesta se termină cu un caracter de forma $, % sau #) unde se pot introduce comenzi. Execuția unei comenzi înseamnă, de obicei, generarea unui proces dintr-un executabil asociat comenzii.

O deprindere utilă în lucrul cu comenzile este prelucrarea șirului de caractere care le formează. Astfel, un utilizator avansat va edita comenzile sale și va folosi istoricul pus la dispoziție de un shell pentru a fi cât mai eficient. Shell-ul pune la dispoziția utilizatorului mai multe facilități de editare și utilizare rapidă a comenzilor.

12.2.1 Editarea comenzilor

O comandă este introdusă la promptul shell-ului prin folosirea tastelor corespunzătoare, după care se apasă ENTER și comanda va fi executată.

Există, însă, situații în care se greșește sau se uită un argument sau o literă și este nevoie de editarea acelei comenzi pentru obținerea efectului dorit. Editarea comenzii poate însemna adăugarea sau șteregerea unui caracter, a unui set de caractere, a unui argument al comenzii. În mod evident, la fel ca în cazul unui editor, este nevoie și de parcurgerea comenzii scrise.

Editarea comenzilor în Bash este realizată prin intermediul bibliotecii *readline*. Detalii despre aceasta găsiți la online[1]. Biblioteca *readline* este cea care asigură editarea unei comenzi, oferind interfața de control a terminalului.

Comenzile puse la dispoziție de readline pentru editarea comenzilor shell sunt inspirate din editorul **Emacs**. Urmând convenția **Emacs**, vom prescurta cu `C` tasta `Control` iar cu `M` tasta `Alt` (`Meta`). Prezentăm, în continuare, câteva dintre cele mai utile comenzi:

- `C-f`, `C-b` – un caracter înainte, respectiv înapoi (echivalent cu `Săgeată-Dreapta` și `Săgeată-Stânga`)
- `C-a`, `C-e` – început de linie, respectiv sfârșit de linie (echivalent cu `Home` și `End`)
- `M-f`, `M-b` – cuvânt înainte, respectiv înapoi; pe unele terminale (spre exemplu `gnome-terminal`) trebuie dezactivate scurtăturile terminalului pentru a permite comenzile *readline*
- `C-d` – șterge caracterul de sub cursor (echivalent cu `Delete`)
- `Backspace` – șterge caracterul de dinainte de cursor
- `M-d` – șterge până la sfârșitul cuvântului
- `M-Backspace` – șterge până la începutul cuvântului
- `C-k` – șterge până la sfârșitul liniei
- `C-w` – șterge până la începutul liniei
- `C-y` – lipește (*paste*, *yank*) ceea ce s-a șters ultima oară

[1] http://www.gnu.org/software/bash/manual/html_node/Readline-Interaction.html

- `C-_` (`Control-underscore`) – anulează (*undo*) ultima comandă de editare.

Pentru cei ce doresc acest lucru, editarea comenzilor folosind *readline* poate fi personalizată folosind un fișier de configurare. Pentru mai multe detalii consultați manualul[1].

12.2.2 Folosirea istoricului. Completare automată

Biblioteca *readline* memorează comenzile folosite în shell într-un istoric care poate fi folosit ulterior. Se pot folosi comenzi *readline* pentru parcurgerea comenzilor shell. Astfel `C-p` sau `C-n` parcurg comenzile anterioare sau ulterioare comenzii curente. Sunt echivalente cu `Săgeată-Sus` și `Săgeată-Jos`.

Istoricul este salvat la părăsirea shell-ului într-un fișier. În cazul **`bash`** acesta este `~/.bash_history`. La o nouă autentificare, shell-ul va încărca respectivele comenzi din history și utilizatorul le va putea reutiliza.

Căutarea comenzilor (*reverse search*)

În afara parcurgerii comenzilor din istoric, o opțiune foarte utilă este căutarea acestora. Pentru căutarea comenzilor în istoric se poate folosi comanda `C-r` (*reverse search*):

```
1 razvan@anaconda:~$ cd
2 (reverse-i-search)'chm': chmod o+w uploads/
```

În exemplul de mai sus, apăsarea `C-r` duce la schimbarea promptului. Utilizatorul poate introduce un șir de căutare pentru identificarea unei comenzi. Pentru fiecare caracter introdus, shell-ul va afișa prima comandă anterioră cea mai potrivită. Completarea șirului va duce la refacerea căutării. În afara introducerii șirului de căutare există câteva comenzi utile în modul de căutare:

- dacă se apasă (din nou) `C-r` se caută următoarea potrivire;
- dacă se apasă `ESC` sau `C-j` se încheie căutarea; comanda afișată la prompt va fi cea găsită;
- dacă se apasă `ENTER` se încheie căutarea și se execută comandă găsită;
- dacă se apasă `C-g` se întrerupe căutarea cu revenire la comanda de dinainte de căutare.

Completare automată (*autocompletion*)

O facilitate extrem de utilă în eficientizarea lucrului cu comenzile shell este cea de completare automată (*autocompletion*). Această facilitate înseamnă introducerea unui șir parțial de început al unei comenzi sau al unei intrări în sistemul de fișiere și apoi apăsarea tastei `TAB` pentru completarea automată. Spre exemplu, dacă cineva dorește să verifice server-ul de nume (DNS) al unui sistem, va folosi comanda:

[1] http://www.gnu.org/software/bash/manual/html_node/Readline-Init-File.html

```
ubuntu@ubuntu:~$ cat /etc/resolv.conf
```

Un calcul simplu arată că este nevoie de apăsarea a 21 de taste (cu ENTER) pentru execuția comenzii.

Putem folosi, în schimb, completarea automată și vom apăsa tastele în mod prezentat mai jos:

```
ubuntu@ubuntu:~$ cat /e<TAB>res<TAB>.<TAB><ENTER>
```

Un calcul la fel de simplu indică apăsarea a doar 14 taste (plus că tasta `TAB` devine foarte importantă și este apăsată mult mai ușor). Comanda afișată la prompt va fi identică cu cea de mai sus și, drept urmare, efectul va fi același.

În consecință, se recomandă insistent folosirea facilității de completarea automată. Această facilitate nu este prezentă numai la shell-urile unui sistem de operare ci și pentru comanda dispozitivelor de rețea și a altor echipamente care dispun de o interfață în linia de comandă. În plus, folosirea tastei `TAB` are un rol important în evitarea erorilor de scriere în cazul intrărilor sistemului de fișiere. Experiența personală a autorilor acestei cărți a surprins, nu de puține ori, crearea de fișiere cu nume "apropiate", dar nu identice, de cele reale (precum `/etc/rezolv.conf`, `/etc/resolv.comf`, `/etc/resolv.cf` etc.). Aceste erori ar fi putut fi evitate (dincolo de o mai mare viteză de utilizare a comenzilor) prin folosirea facilității de completare automată.

În situația în care există mai multe posibilități de completare, apăsarea tastei `TAB` nu are niciun efect. Totuși, se pot vizualiza toate aceste combinații prin apăsarea de două ori a tastei `TAB`. Astfel, apăsarea `z<TAB><TAB>` va duce la afișarea tuturor comenzilor care încep cu litera z:

```
razvan@anaconda:~$ z<TAB><TAB>
zcat         zegrep       zforce       zipgrep      zless
zcmp         zeisstopnm   zgrep        zipinfo      zmore
zdiff        zenity       zip          zipnote      znew
zdump        zfgrep       zipcloak     zipsplit     zsoelim
```

La fel, folosirea comenzii **`cat /<TAB><TAB>`** duce la afișarea tuturor intrărilor care se găsesc în directorul rădăcină al sistemului de fișiere:

```
razvan@anaconda:~$ cat /
.viminfo         home/            mnt/             tmp/
aquota.user      initrd/          opt/             usr/
bin/             initrd.img       proc/            var/
boot/            initrd.img.old   root/            vmlinuz
cdrom/           lib/             sbin/            vmlinuz.old
dev/             lost+found/      srv/
etc/             media/           sys/
```

Extinderea completării automate După cum s-a observat, completarea automată se referă la comenzi și la intrări în sistemul de fișiere. Totuși, aceasta poate fi extinsă prin intermediul unor scripturi personalizate la argumentele posibile ale unei comenzi.

Astfel fișierul `/etc/bash_completion` este folosit pentru extinderea completării automate. Pentru folosirea acestui script este necesară rularea comenzii:

```
ubuntu@ubuntu:~$ . /etc/bash_completion
```

Câteva linii din acest fișier:

```
1 complete -f -X '!*.@(sxi|sti|pps|ppt|pot|odp|otp)' ooimpress
```

Linia de mai sus înseamnă că dacă un utilizator folosește în linia de comandă **ooimpress**, argumentele posibile ale acesteia vor fi considerate doar fișierele cu extensiile precizate.

```
1 # user commands see only users
2 complete -u su usermod userdel passwd chage write chfn groups slay w
```

Linia de mai sus permite ca argumente numai nume de utilizatori pentru comenzile de lucru cu aceștia.

În afara extinderilor existente în `/etc/bash_completion`, utilizatorul își poate defini noi extinderi pentru a le integra în shell. Se recomandă ca acestea să fie plasate în directorul `/etc/bash_completion.d`. Generarea de noi extensii de completare depășește domeniul de interes al acestei cărți. Recomandăm celor interesați pagina web asociată proiectului [1] sau articolul legat de completarea automata de pe *Debian Administration*[2].

12.2.3 Comenzi interne (*built-in*) și comenzi externe

Comenzile care pot fi rulate dintr-un shell sunt de două tipuri:

1. **comenzi externe**: sunt rulate prin intermediul unui executabil
2. **comenzi interne**: sunt executate direct de shell fără invocarea unui alt program.

Comenzile interne (sau *built-in*) sunt utilizate pentru implementarea de facilități care sunt fie imposibil fie neconvenabil de realizat folosind programe separate. Un bun exemplu este comanda **cd** de schimbare a directorului curent. Această comandă nu poate fi folosită dintr-un executabil extern, întrucât la sfârșitul rulării comenzii s-ar pierde orice efect ar avea (și nu ar fi posibilă, astfel, schimbarea persistentă a directorului curent).

Exemple de comenzi externe și executabilul folosit pentru rularea acestora sunt:

- **ls** cu `/bin/ls`
- **file** cu `/usr/bin/file`
- **su** cu `/bin/su`
- **ifconfig** cu `/sbin/ifconfig`

Exemple de comenzi interne sunt:

- `:` – comanda nu face nimic, doar expandează variabile și realizează redirectările; astfel, rularea comenzii:

```
1 ubuntu@ubuntu:~$ : > out.txt
```

duce la trunchierea fișierului out.txt. Versiunea și mai simplă este

```
1 ubuntu@ubuntu:~$ > out.txt
```

[1] http://www.caliban.org/bash/index.shtml
[2] http://www.debian-administration.org/articles/316

- `.` – comanda este folosită pentru interpretarea unui fișier (de obicei script) în shell-ul curent. Echivalentă este comanda **`source`**. Un exemplu de folosire este încărcarea configurațiilor din `/etc/bash_completion`:

```
ubuntu@ubuntu:~$ . /etc/bash_completion
```

sau

```
ubuntu@ubuntu:~$ source /etc/bash_completion
```

- **`cd`** – comanda folosită pentru schimbarea directorului curent
- **`echo`** – afișarea șirului de caractere primit ca parametru la ieșirea standard
- **`alias`/`unalias`** – configurarea/dezactivarea configurării unui alias pentru o comandă. Astfel, dacă folosim:

```
alias mygrep='grep --color=tty -d skip'
```

comanda **`mygrep`** (alias) va fi lansată ca o comandă **`grep`** cu argumentele precizate; se poate folosi ca alias chiar numele comenzii, în cazul în care utilizatorul dorește rularea implicită cu acele argumente:

```
alias grep='grep --color=tty -d skip'
```

De notat este faptul că o comandă poate fi implementată atât în shell (*built-in*) cât și prin intermediul unui executabil. Astfel de comenzi sunt **`echo`** și **`time`**. În aceste situații comanda executată implicit este comanda internă. Pentru execuția comenzii externe trebuie apelată calea completă. În exemplul de mai jos sunt prezentate cele două tipuri de rulări pentru comanda **`time`** (internă și externă):

```
razvan@anaconda:~/code$ time ls
asm  boot_hello  latex  stack_ovfl  static  tests

real    0m0.004s
user    0m0.000s
sys     0m0.000s

razvan@anaconda:~/code$ /usr/bin/time ls
asm  boot_hello  latex  stack_ovfl  static  tests
0.00user 0.00system 0:00.00elapsed 133%CPU (0avgtext+0avgdata 0
maxresident)k
0inputs+0outputs (0major+254minor)pagefaults 0swaps
```

Se observă că poate exista o deosebire între formatele de ieșire ale unei comenzi interne, respectiv externe.

De asemenea, mecanismele de ajutor folosite pentru comenzi (**`whereis`**, **`apropos`**, `--help`, **`man`**, **`info`**) funcționează doar pentru comenzi externe. Pentru comenzi interne va trebui consultată documentația de *Bash* (**`man bash`**, **`info bash`**) sau folosită comanda (internă) **`help`**:

```
razvan@anaconda:~/code$ help time
time: time [-p] PIPELINE
    Execute PIPELINE and print a summary of the real time, user CPU time
,
    [...]

razvan@anaconda:~/code$ help alias
```

```
7 alias: alias [-p] [name[=value] ... ]
8      'alias' with no arguments or with the -p option prints the list
9     [...]
```

O listă completă a comenzilor interne *Bash* găsiți în pagina de manual. (**man bash**/**info bash** sau online[1])

12.2.4 Execuția unei comenzi shell

După cum s-a precizat, comenzile sunt de două tipuri: interne și externe. Clasificarea este dată de modul în care comenzile se execută în cadrul shell-ului. Astfel, comenzile interne (*built-in*) sunt executate direct în cadrul shell-ului, altfel spus în cadrul procesului curent. De partea cealaltă, execuția unei comenzi externe înseamnă crearea unui proces fiu în shell-ul curent și înlocuirea imaginii procesului cu executabilul asociat comenzii. Mai multe detalii despre caracteristicile și crearea unui proces se pot găsi în secțiunea 5.2.

Un proces astfel creat va conține ca imagine executabilul asociat comenzii. De obicei, shell-ul va aștepta terminarea procesului curent după care va oferi promptul utilizatorului. Acest efect poate fi schimbat prin rularea procesului în fundal (*background*) cu ajutorul operatorului & (vezi secțiunea 5.3).

De reținut este faptul că, în cazul înlănțuirii mai multor comenzi (vezi secțiunea 5.5.1), se creează un proces pentru fiecare comandă. Astfel, la rularea comenzii:

```
1 ubuntu@ubuntu:~$ ls -l | sort
```

se vor crea două procese: unul generat din executabilul **/bin/ls** iar altul generat din executabilul **/usr/bin/sort**. Shell-ul este procesul părinte al celor două procese și va asigura comunicația între acestea (ieșirea comenzii **ls -l** va fi redirectată către intrarea comenzii sort).

Figura 12.1 ilustrează grafic execuția comenzii de mai sus.

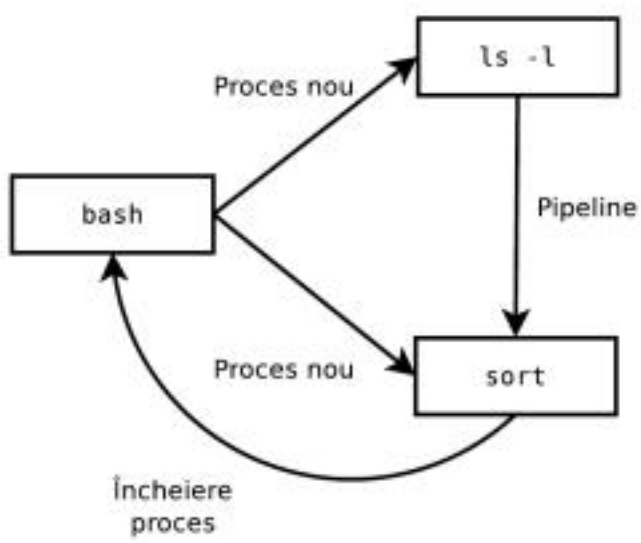


Figura 12.1: Execuția unei comenzi în shell

[1] http://www.gnu.org/software/bash/manual/bashref.html#SEC55

12.3 Scripturi shell

După cum am specificat, un script shell este un fișier text (un program) conținând comenzi shell, variabile, structuri de control și funcții cu rolul de rezolvare a unei probleme. Un script shell este interpretat de shell. Interpretarea înseamnă parcurgerea fiecărei înlănțuiri de comenzi sau structuri de control și realizarea sarcinilor descrise de aceasta.

Ceea ce separă un script shell de limbajele interpretate obișnuite este folosirea directă a comenzilor shell, a facilităților de înlănțuire a comenzilor puse la dispoziție și a fișierelor de configurare specifice sistemului.

Comenzile componente ale unui script shell pot fi executate manual la promptul shell-ului cu aceleași rezultate ca în cazul rulării scriptului. Astfel, liniile următoare dintr-un script shell:

```
1 if test 10 -gt 4; then
2   echo "mai mare"
3 else
4   echo "mai mic"
5 fi
```

pot fi executate direct în linia de comanda:

```
1 razvan@anaconda:~$ if test 10 -gt 4; then echo "mai mare"; else echo "mai
   mic"; fi
2 mai mare
```

sau astfel:

```
1 razvan@anaconda:~$ if test 10 -gt 4
2 > then
3 > echo "mai mare"
4 > else
5 > echo "mai mic"
6 > fi
7 mai mare
```

În ultimul exemplu, shell-ul oferă utilizatorului un miniprompt (>) pentru a indica acestuia că nu a finalizat comanda. În cazul de față comanda este `if`. Detalii despre `if` și despre efectul acestei bucăți de script se regăsesc în secțiunea 12.4.3.

12.3.1 Cel mai simplu script shell

Pentru a păstra tonul tutorialelor de limbaje de programare, cel mai simplu script shell pe care îl vom descrie va fi cel care va afișa mesajul "Hello, World!" la ieșirea standard. Programul este prezentat mai jos:

```
1 echo "Hello, World!"
2 exit 0
```

Programul este foarte simplu. Folosim comanda **`echo`** pentru a afișa un șir de caractere la ieșirea standard și comanda **`exit`** pentru ca programul să se termine cu succes.

Folosirea comenzii **exit** este opțională; un script shell va întoarce în mod implicit 0 dacă se ajunge la sfârșitul programului. Astfel, scriptul nostru shell poate fi scris într-o singură linie:

```
1 echo "Hello, World!"
```

Rularea unui script shell

Rularea unui script shell se realizează la fel ca în cazul unui limbaj interpretat prin transmiterea sa ca parametru interpretorului:

```
1 razvan@anaconda:~/uso/scripting$ bash hw.bash
2 Hello, World!
```

Scriptul shell poartă denumirea de `hw.bash`. Se recomandă ca scripturile shell Bash să aibă extensia `.sh` sau `.bash`, cu toate că în lumea Unix extensia poate fi omisă. Se observă afișarea mesajului "Hello, World!" la ieșirea standard.

În spatele acestei execuții, shell-ul generează un nou proces care interpretează conținutul scriptului primit ca parametru.

#! – shebang

Mecanismul uzual de execuție a unui shell script este acela prin care se execută un program obișnuit:

```
1 ubuntu@ubuntu:~$ ./hw.bash
```

Totuși, pentru a putea realiza acest lucru avem de îndeplinit două precondiții:

1. scriptul trebuie să fie executabil
2. în momentul execuției trebuie să se cunoască ce interpretor va fi folosit (`/bin/bash` pentru Bash, `/usr/bin/python` pentru Python, `/usr/bin/perl` pentru Perl etc.)

Primul pas se realizează foarte simplu. Se adaugă drepturi de execuție folosind comanda chmod:

```
1 razvan@anaconda:~/uso/scripting$ chmod a+x hw.bash
2
3 razvan@anaconda:~/uso/scripting$ ls -l hw.bash
4 -rwxr-xr-x 1 razvan razvan 42 Aug 31 18:46 hw.bash
```

Pentru realizarea celui de-al doilea pas se folosește perechea de caractere "`#!`" (diez și semnul exclamării) denumită și *shebang*. Această pereche de caractere este urmată de calea completă către programul care va interpreta scriptul. Linia care conține shebang și calea completă către interpretor este prima linie din script. În momentul în care scriptul este lansat în execuție, shell-ul parcurge prima linie și folosește interpretorul specificat pentru a executa scriptul. Exemple de linii *shebang* sunt:

- `#!/bin/bash` – executarea scriptului folosind Bash
- `#!/usr/bin/perl` – executarea scriptului folosind Perl

- `#!/usr/bin/php` – executarea scriptului folosind PHP
- `#!/usr/bin/python` – executarea scriptului folosind Python

Astfel, după adăugarea drepturilor de execuție și a liniei *shebang* scriptului nostru:

```
1 razvan@anaconda:~/uso/scripting$ cat hw.bash
2 #!/bin/bash
3
4 echo "Hello, World!"    # afisare mesaj
5 exit 0
```

îl putem executa:

```
1 razvan@anaconda:~/uso/scripting$ ./hw.bash
2 Hello, World!
```

Un script shell nu necesită prezența liniei *shebang*. În mod implicit un script este interpretat folosind **`/bin/bash`**. Se recomandă, totuși, prezența acestei linii pentru a asigura consistența.

În prezent, distribuția Debian și distribuțiile derivate (precum Ubuntu, Kubuntu etc.) folosesc ca shell *Bash*, iar ca interpretor implicit de script-uri shell *Dash*. Interpretorul de script-uri shell implicit este **`/bin/sh`** (link simbolic către *Dash*).

Dash este mult mai rapid și mai eficient din punct de vedere al consumului de resurse dar este și mai limitat ca și sintaxă, comparativ cu *Bash*. Așadar, atunci când modificați scripturi existente în sistem care au în linia *shebang* ca interpretor **`/bin/sh`**, luați în considerare că sintaxa este limitată[1] comparativ cu *Bash*.

12.3.2 Comentarii într-un script shell

Comentariile într-un script shell se realizează cu ajutorul caracterului `#`. Un comentariu începe de la apariția `#` până la sfârșitul liniei, asemănător cu `//` din C++. Exemple de comentarii sunt:

```
1 # comentariu
2 echo "hello"  # comentariu
3 # echo "hello"  (comanda comentata)
```

12.3.3 Comenzi simple shell

În mod evident, scriptul shell prezentat anterior nu este un script shell în adevăratul sens al cuvântului. Aceasta deoarece același efect poate fi obținut prin rularea simplă a comenzii echo urmată de argumentul "Hello, World!" la promtpul shell-ului. Un script shell va conține mai multe comenzi care vor fi interpretate într-o singură instanță în momentul execuției scriptului.

Scripturile shell folosesc de multe ori comenzi interne. Vom prezenta, în continuare, o parte din cele mai folositoare comenzi interne:

[1] https://wiki.ubuntu.com/DashAsBinSh

echo

Comanda **echo** permite afișarea șirului primit ca parametru la ieșirea standard a terminalului:

```
razvan@anaconda:~$ echo "test"
test
```

Opțiuni utile pentru **echo** sunt:

- `-n`: dezactivează afișarea implicită a unui caracter newline la sfârșitul liniei:

```
razvan@anaconda:~$ echo -n "test"
testrazvan@anaconda:~$
```

- `-e`: permite interpretarea caracterelor speciale:

```
razvan@anaconda:~$ echo -ne "test\n\t\ttest2\n"
test
                test2
```

printf

Comanda **printf** permite, ca și **echo**, afișarea unui șir de caractere la ieșirea standard. Permite, însă, la fel ca funcția `printf` din C, formatarea șirului de afișare:

```
razvan@anaconda:~$ printf "%s    %03d\n" "test" 20
test    020
```

Se observă similaritatea cu funcția printf din C: primul argument este șirul de formatare iar celelalte sunt argumentele de afișat la ieșirea standard.

exit

Comanda **exit** permite întreruperea execuției scriptului. Comanda poate primi un argument care specifică valoarea de retur. Această valoare este implicit 0.

export

Comanda **export** permite exportarea unei variabile (sub forma unei variabile de mediu) din scriptul curent în shell. Astfel, după ce shell-ul a încheiat interpretarea scriptului, variabila va putea fi folosită direct din shell. Pentru mai multe detalii în legătură cu variabile în script-uri consultați secțiunea 12.4.1.

set

Comanda **set** este utilizată pentru configurarea și personalizarea shell-ului. O opțiune utilă este opțiunea `-x` care permite afișarea comenzilor unui script shell pe măsură ce acestea sunt executate.

read

Comanda **read** este folosită pentru a citi date de la intrarea standard și pentru stocarea acestora într-o variabilă. În exemplul de mai jos, se dorește stocarea datelor introduse de utilizator în variabila `a`:

```
1 razvan@anaconda:~/uso/scripting$ read a
2 test_msg
3
4 razvan@anaconda:~/uso/scripting$ echo $a
5 test_msg
```

Se observă că variabila conține șirul *test_msg* introdus de utilizator.

12.3.4 One liners

Multe din problemele ce necesită utilizarea facilităților shell pot fi realizate prin intermediul unei simple linii. S-a observat că nu este necesar (și nici eficient) să utilizăm un script shell doar pentru afișarea unui mesaj precum "Hello, World!\n" la ieșirea standard.

Folosind redirectări sau înlănțuirea comenzilor se pot obține efecte imediate într-o singură linie de shell. Astfel de comenzi combinate poartă numele de **one liners**.

Redirectarea intrării/ieșirii unei comenzi

Redirectarea unei comenzi se oferă la posibilitatea de substituire a intrării, ieșirii sau erorii standard a unei comenzi cu un fișier. Astfel, redirectarea intrării standard înseamnă că o comandă va citi informații dintr-un fișier în locul citirii de la intrarea standard. Redirectarea ieșirii standard înseamnă că rezultatul comenzii se va regăsi într-un fișier de ieșire.

Pentru redirectarea intrării standard se folosește operatorul < (caracterul "mai mic") urmat de numele fișierul din care se realizează redirectarea. În cazul redirectării ieșirii standard, operatorul folosit este > (mai mare). Informații despre redirectarea intrării și ieșirii standard se regăsesc și în secțiunea 4.4.

Vom exemplifica redirectarea folosind utilitarul **cat**. Comanda **cat** realizează, în mod implicit, afișarea conținutului intrării standard la ieșirea standard. Un exemplu de rulare este prezentat în continuare:

```
1 razvan@anaconda:~$ cat
2 mesaj1
3 mesaj1
4 mesaj2
5 mesaj2
```

Pentru încheierea introducerii de informații de la intrarea standard trebuie folosită combinația `CTRL-D`, însemnând sfârșit de fișier (*end of file*). Se observă că ceea ce introduce utilizatorul la intrarea standard se afișează la ieșirea standard.

Dacă vom redirecta un fișier la intrarea standard, efectul va fi afișarea conținutului fișierului la ieșirea standard:

```
razvan@anaconda:~/uso/scripting$ cat < hw.bash
#!/bin/bash

echo "Hello, World!"

exit 0
```

Acest lucru este echivalent cu transmiterea fișierului ca argument comenzii **cat**.

În mod similar, dacă redirectăm ieșirea standard într-un fișier, acesta va conține ceea ce a introdus utilizatorul la intrarea standard (se folosește `CTRL-D` pentru încheiere):

```
razvan@anaconda:~/uso/scripting$ cat > out.txt
introducem mesaj
dupa care apasam CTRL-D

razvan@anaconda:~/uso/scripting$ cat out.txt
introducem mesaj
dupa care apasam CTRL-D
```

În exemplul de mai sus, comanda **cat** a afișat fișierul `out.txt` prin transmiterea acestuia ca argument, echivalent cu redirectarea acestuia către intrarea standard a comenzii. Acest comportament se extinde și la alte comenzii de prelucrare de fișiere text (vezi secțiunea 12.5).

Un efect care poate părea surprinzător este folosirea simultană a redirectării intrării și ieșirii standard. În acest fel, conținutul fișierului de la intrare este copiat în fișierul de la ieșire. Efectul este identic cu folosirea comenzii **cp**:

```
razvan@anaconda:~/uso/scripting$ cat < hw.bash > new.bash

razvan@anaconda:~/uso/scripting$ cat new.bash
#!/bin/bash

echo "Hello, World!"

exit 0
```

Pentru redirectarea erorii standard se folosește operatorul `2>`. Aceasta are legătură cu descriptorul de fișier asociat: intrarea standard are asociat descriptorul de fișier `0`, ieșirea standard are asociat descriptorul de fișier `1` iar eroarea standard are asociat descriptorul de fișier `2`.

Astfel, dacă se doresc redirectate într-un fișier erorile și avertismentele emise de **gcc** se folosește operatorul `2>`:

```
razvan@anaconda:~/uso/scripting$ gcc hw.c 2> err_warn.txt

razvan@anaconda:~/uso/scripting$ cat err_warn.txt
hw.c: In function 'main':
hw.c:3: warning: incompatible implicit declaration of built-in function '
printf'

razvan@anaconda:~/uso/scripting$ gcc hw.c 2> /dev/null
```

În cel de-al doilea exemplu, s-a redirectat eroarea standard către `/dev/null`. Aceasta înseamnă că nu se dorește afișarea la eroarea standard sau într-un fișier a diverselor informații de avertisment afișate de **gcc**.

Se poate redirecta atât ieșirea standard cât și eroarea standard folosind operatorul `&>` (Atenție: acest operator nu funcționeaza în interpretorul shell *Dash*[1]). În exemplu de mai jos se redirectează atât ieșirea standard cât și eroarea standard a execuției comenzii **`strace`**:

```
1 razvan@anaconda:~/uso/scripting$ strace ls &> out_ls.txt
```

Operatorul >> Operatorul `>>` este utilizat pentru redirectarea ieșirii standard însă cu adăugarea (*append*) a informațiilor redirectate la sfârșitul fișierului. Echivalent există operatorul `2>>` pentru adăugarea informațiilor de la eroarea standard. În exemplul de mai jos scriem ieșirile comenzilor **`ls`**, **`ps`** și **`uptime`** într-un același fișier:

```
1  razvan@anaconda:~/uso/scripting$ ls > out.txt
2
3  razvan@anaconda:~/uso/scripting$ ps >> out.txt
4
5  razvan@anaconda:~/uso/scripting$ uptime >> out.txt
6
7  razvan@anaconda:~/uso/scripting$ cat out.txt
8  a.out
9  err_warn.txt
10 hw.bash
11 hw.c
12 new
13 new.bash
14 out.txt
15 out_ls.txt
16   PID TTY          TIME CMD
17 30624 pts/2    00:00:00 bash
18 31508 pts/2    00:00:00 ps
19  15:34:59 up 59 days, 39 min,  2 users,  load average: 0.37, 0.55, 0.61
```

Here documents; operatorul << Operatorul `<<` este folosit în ceea ce se cheamă **here document**. Un *here document* este folosit, de obicei, cu utilitare interactive. Formatul de utilizare este:

```
1 command <<word
```

Acest lucru va impune comenzii citirea informației aflată în continuarea comenzii de la intrarea standard până la întâlnirea cuvântului *word* în corpul scriptului. În exemplul de mai jos, se citește de la intrarea standard până la întâlnirea cuvântului *END*:

```
1  razvan@anaconda:~/uso/scripting$ cat here.bash
2  #!/bin/bash
3
4  cat <<END
5  mesaj simplu
6  ... pentru un here document ...
7
8  avantajul este ca nu trebuie sa folosesc multe comenzi echo
9  END
10
11 echo "s-a terminat"
12 razvan@anaconda:~/uso/scripting$ bash here.bash
```

[1] https://wiki.ubuntu.com/DashAsBinSh

```
mesaj simplu
... pentru un here document ...

avantajul este ca nu trebuie sa folosesc multe comenzi echo
s-a terminat
```

După cum se observă și din comentariile fișierului, un *here document* poate fi folosit în locul comenzii **echo**. Un exemplu util îl constituie generarea unei pagini web simple. O primă alternativă este folosirea comenzii **echo**:

```
razvan@ragnarok:~/uso/scripting$ cat echo.bash
#!/bin/bash

hostname="ragnarok"
name="razvan"

echo -e "<html>"
echo -e "\t<title>"
echo -e "\t\tPagina mea"
echo -e "\t</title>"
echo -e "\t<body>"
echo -e "\t\t<h3>Statia este $hostname</h3>"
echo -e "\t\t<h3>Numele meu este $name</h3>"
echo -e "\t\t<p>Data este $(date)</p>"
echo -e "\t</body>"
echo -e "</html>"

razvan@ragnarok:~/uso/scripting$ bash echo.bash
<html>
        <title>
                Pagina mea
        </title>
        <body>
                <h3>Statia este ragnarok</h3>
                <h3>Numele meu este razvan</h3>
                <p>Data este Tue Sep  4 23:23:05 EEST 2007</p>
        </body>
</html>
```

Se observă că este incomodă folosirea comenzii **echo**. O soluție este utilizarea unui *here document*:

```
razvan@ragnarok:~/uso/scripting$ cat here.bash
#!/bin/bash

hostname="ragnarok"
name="razvan"

cat <<EndOfHtml
<html>
        <title>
                Pagina mea
        </title>
        <body>
                <h3>Statia este $hostname</h3>
                <h3>Numele meu este $name</h3>
                <p>Data este $(date)</p>
        </body>
</html>
EndOfHtml
```

Informații suplimentare despre *here documents* se pot găsi online[1].

Here strings; operatorul <<< Operatorul <<< permite folosirea de *here strings*, adică redirectarea unui șir de caractere către intrarea standard a unei comenzi. În exemplul de mai jos, se folosește comanda **read** pentru a citi conținutul unui șir într-un set de variabile:

```
razvan@anaconda:~$ read a b c d <<< "alfa beta gamma delta "

razvan@anaconda:~$ echo $a $b $c $d
alfa beta gamma delta
```

Ca și în cazul operatorului <<, același rezultat poate fi obținut prin folosirea comenzii **echo**:

```
razvan@anaconda:~$ echo "alfa beta gamma delta" | read a b c d
```

Trunchierea unui fișier **Trunchierea unui fișier** înseamnă eliminarea conținutului acestuia: fișierul devine gol (dimensiune 0). Trunchierea unui fișier se realizează prin redirectarea conținutului `/dev/null` în fișier. Acest lucru se realizează folosind comanda:

```
razvan@anaconda:~/uso/scripting$ ls -l out.txt
-rw-r--r-- 1 razvan razvan 220 Sep  4 15:34 out.txt

razvan@anaconda:~/uso/scripting$ cat /dev/null > out.txt

razvan@anaconda:~/uso/scripting$ ls -l out.txt
-rw-r--r-- 1 razvan razvan 0 Sep  4 15:35 out.txt
```

Mai simplu, trunchierea unui fișier se realizează prin redirectarea unei comenzi care nu afișează nimic la ieșirea standard. O astfel de comandă este : (comanda care nu face nimic):

```
razvan@anaconda:~/uso/scripting$ : > out.txt
```

Şi mai simplu, trunchierea se realizează folosind comanda de mai jos:

```
razvan@anaconda:~/uso/scripting$ > out.txt
```

Comanda este echivalentă cu cea de mai sus, cu absența comenzii :.

Înlănțuirea comenzilor

Shell-ul pune la dispoziția utilizatorului operatori care permit înlănțuirea diverselor comenzi pentru obținerea de noi funcționalități. Dintre aceștia, cel mai cunoscut este **operatorul** | (*pipe*) care redirectează ieșirea unei comenzi la intrarea alteia. Detalii despre acest operator se găsesc și în capitolul 5.

[1] http://tldp.org/LDP/abs/html/here-docs.html

Operatorul | (pipe) Un exemplu de utilizare al operatorului | este căutarea unei informații într-un fișier. O comandă uzuală este:

```
ubuntu@ubuntu:~$ cat file.txt | grep keyword
```

Căutarea informației în fișier se realizează cu ajutorul comenzii **grep**, descrisă în secțiunea 12.5.7.

Un exemplu mai complex care utilizează o serie de comenzi prezentate în acest capitol este determinarea numărului de utilizatori unici care s-au autentificat în sistem într-o zi dată. Comenzile utilizate sunt **last**, **grep**, **cut**, **sort**, **uniq**, **wc**. Detalii despre aceste vor fi prezentate în secțiunile următoare.

Pentru început, vom folosi comanda **last** care afișează autentificările în sistem. Comanda **grep** este folosită pentru a extrage numai acele linii care conțin ziua dorită:

```
razvan@anaconda:~/uso/scripting$ last -30 | grep Mon
sergiu   pts/4        89.18.20.56      Mon Sep  3 23:36 - 23:46  (00:09)
razvan   pts/4        dhcp-204.cs.pub. Mon Sep  3 23:19 - 23:35  (00:15)
carpalex pts/0        86.121.140.140   Mon Sep  3 23:14 - 01:41  (02:27)
alexef   pts/0        89.120.196.59    Mon Sep  3 22:42 - 22:51  (00:08)
cojocar  pts/0        86.127.17.156    Mon Sep  3 19:50 - 19:51  (00:01)
ddvlad   pts/3        84.247.45.23     Mon Sep  3 16:00 - 16:01  (00:00)
root     pts/0        dhcp-204.cs.pub. Mon Sep  3 15:14 - 17:33  (02:18)
ddvlad   pts/0        84.247.45.23     Mon Sep  3 15:07 - 15:09  (00:01)
valentin pts/2        86.122.195.234   Mon Sep  3 13:47 - 02:43  (12:55)
alexef   pts/0        89.120.196.59    Mon Sep  3 13:30 - 15:04  (01:34)
razvan   pts/0        dhcp-204.cs.pub. Mon Sep  3 10:30 - 10:44  (00:13)
stefanb  pts/0        adsl-76-200-147- Mon Sep  3 10:05 - 10:17  (00:12)
stefanb  pts/0        adsl-76-200-147- Mon Sep  3 06:13 - 07:57  (01:43)
```

După obținerea autentificărilor în sistem vrem să extragem numai numele utilizatorilor. Pentru aceasta vom folosi comanda **cut**:

```
razvan@anaconda:~/uso/scripting$ last -30 | grep Mon | cut -d' ' -f1
sergiu
razvan
carpalex
alexef
cojocar
ddvlad
root
ddvlad
valentin
alexef
razvan
stefanb
stefanb
```

În acest moment avem numele utilizatorilor care s-au autentificat în sistem în ziua de *Luni*. Mai departe, va trebui să extragem o singură instanță a numelor care se repetă. Drept urmare vom folosi utilitarele **sort** și **uniq**:

```
razvan@anaconda:~/uso/scripting$ last -30 | grep Mon | cut -d' ' -f1 |
sort | uniq
alexef
carpalex
cojocar
ddvlad
```

```
6  razvan
7  root
8  sergiu
9  stefanb
10 valentin
```

Mai rămâne să afișăm numărul de autentificări. Acest lucru se realizează prin contorizarea liniilor obținute la pasul anterior, cu ajutorul utilitarului **wc**:

```
1  razvan@anaconda:~/uso/scripting$ last -30 | grep Mon | cut -d' ' -f1 |
   sort | uniq | wc -l
2  9
```

Comanda de mai sus poate da rezultate incorecte în cazul în care `Mon` mai apare altundeva în rezultatul furnizat de **`last`**. Dacă ar exista un utilizator `Monica`, atunci vor apărea și liniile ce descriu autentificarea în sistem a utilizatorului `Monica` (chiar dacă autentificarea a avut sau nu loc în ziua de luni). Această problemă va fi corectată în secțiunea 12.11.1.

Operatorii `;`, `||` și `&&` Operatorii `;`, `||` și `&&` de înlănțuire a comenzilor țin cont de valoarea întoarsă de comanda anterioară. Astfel:

- `comm1 ; comm2` – comanda **comm2** se execută după execuția comenzii **comm1**
- `comm1 || comm2` – comanda **comm2** se execută în cazul în care **comm1** se întoarce cu o valoare nenulă
- `comm1 && comm2` – comanda **comm2** se execută în cazul în care **comm1** se încheie cu succes (întoarce 0)

Operatorul `;` este echivalent cu execuția secvențială a fiecărei comenzi.

Operatorul `&&` este folosit când **comm2** depinde de execuția cu succes a comenzii **comm1**. O situație este compilarea din surse și instalarea unei aplicații:

```
1  root@ubuntu:~# make && make install
```

12.4 Programarea shell

După prezentarea câtorva comenzi simple și a operatorilor shell, putem pătrunde în aspectele de programare a unui shell script. După cum s-a precizat, shell-ul pune la dispoziția utilizatorului un limbaj de programare. Acesta conține, ca și alte limbaje de programare, variabile, funcții, structuri de ciclare, instrucțiuni de decizie. Combinate cu celelalte facilități oferite de shell (utilizarea de comenzi deja implementate, înlănțuirea comenzilor, redirectarea), programarea shell devine un instrument puternic la îndemâna utilizatorului.

12.4.1 Variabile

O variabilă are asociat un nume și o valoare. La fel ca în C, numele variabilelor este un șir de caractere ce poate conține litere, cifre sau caracterul _ (*underscore*) și trebuie

să înceapă cu literă sau *underscore*. Exemple de nume de variabile sunt `a`, `a0b`, `a_b`, `AxB_01_D` etc.

Spre deosebire de limbaje precum C, variabilele în shell nu au un tip. Faptul că o variabilă este un număr sau un șir de caractere depinde de contextul în care este folosită. De asemenea, o variabilă poate fi definită oriunde. În mod obișnuit, unei variabile îi este asociată o valoare în momentul definirii. Exemple de inițializări sunt:

```
1 a=0
2 b="alfa"
3 c="a08ss"
4 d=230
```

Un lucru important de reținut este fragilitatea programării shell și constrângerile importante pe care le impune. Una dintre ele este faptul că nu poate exista spațiu înainte sau după semnul = din momentul inițializării unei variabile. Astfel, următoarele inițializări vor fi invalide:

```
1 a =0
2 b = "alfa"
3 d= 230
```

Valoarea unei variabile este referită cu ajutorul simbolului `$`. Astfel, dacă dorim afișarea valorii variabilelor definite mai sus folosim comanda:

```
1 razvan@anaconda:~$ echo $a $b $c $d
2 0 alfa a08ss 230
```

În consecință, dacă dorim inițializarea unei variabile la valoarea unei alteia vom folosi o construcție de forma:

```
1 razvan@anaconda:~$ echo $e
2 0
3
4 razvan@anaconda:~$ f="$b$c"
5
6 razvan@anaconda:~$ echo $f
7 alfaa08ss
```

În ultimul exemplu se observă că variabila `f` conține valorile concatenate ale celorlalte două variabile. De asemenea, se poate observa că ghilimelele nu schimbă sensul simbolului `$`.

Dacă se dorește ca o variabilă să fie un rezultat al unei operații aritmetice cu o altă variabilă se folosește operatorul de expandare aritmetică descris în secțiunea 12.7.1:

```
1 razvan@anaconda:~$ a=4
2
3 razvan@anaconda:~$ c=3
4
5 razvan@anaconda:~$ echo $(($a * $c + 1))
6 13
```

O facilitate importantă a shell-ului este posibilitatea stocării ieșirii unei comenzi într-o variabilă. Acest lucru se realizează prin intermediul construcției `$(command)`. În exemplul de mai jos, reținem în variabila `local_users_num` numărul de utilizatori care au directorul de bază în `/home`:

```
razvan@anaconda:~$ local_users_num=$(cat /etc/passwd | grep /home | wc -l
)

razvan@anaconda:~$ echo $local_users_num
52
```

Dacă vrem să contorizăm și utilizatorul `root`, adăugăm 1 la acea variabilă:

```
razvan@anaconda:~$ total_local=$(($local_users_num + 1))

razvan@anaconda:~$ echo $total_local
53
```

O greșeală frecventă în inițializarea unei variabile este folosirea de spații înainte sau după =. Evitați apariția unor astfel de greșeli în scripturile voastre.

Exemple

- afișarea ultimelor n adrese IP de la care s-au realizat conexiuni pentru un serviciu (inspecție de jurnale)

```
root@anaconda:/home/razvan/uso/scripting# cat -n service_ip.bash
     1  #!/bin/bash
     2
     3  ip_num=3
     4  log_file="/var/log/apache/access.log"
     5
     6  tail -n $ip_num $log_file | cut -d ' ' -f 1

root@anaconda:/home/razvan/uso/scripting# bash service_ip.bash
38.99.44.102
38.99.44.102
38.99.44.102
```

- Afișarea memoriei totale ocupate de primele 3 procese din sistem[1]

```
root@anaconda:/home/razvan/uso/scripting# cat -n mem_proc.bash
     1  #!/bin/bash
     2
     3  rss1=$(ps -e -o rss --sort=-rss | head -n 2 | tail -n 1)
     4  rss2=$(ps -e -o rss --sort=-rss | head -n 3 | tail -n 1)
     5  rss3=$(ps -e -o rss --sort=-rss | head -n 4 | tail -n 1)
     6
     7  total_rss_kb=$(($rss1 + $rss2 + $rss3))
     8
     9  echo "memorie totala ocupata de primele 3 procese:
$total_rss_kb KB"

root@anaconda:/home/razvan/uso/scripting# bash mem_proc.bash
memorie totala ocupata de primele 3 procese: 29088 KB
```

- Adăugarea unui utilizator în sistem în mod neinteractiv

```
root@anaconda:/home/razvan/uso/scripting# cat -n my_add_user.sh
     1  #!/bin/bash
```

[1] presupunem ca nu se modifică procesele pe parcursul execuției script-ului, astfel încât ordinea lor să se păstreze

```
 3      2
 4      3  #
 5      4  # Add a new user; invoke all necesary script and command for
    adding complete
 6      5  # user facilities
 7      6  #
 8      7
 9      8  username="newuser"
10      9  email="razvand@gmail.com"
11     10
12     11  GROUP_NAME="students"
13     12  GROUP_DIR="students"
14     13  QUOTA="250"             # size in megs
15     14
16     15  # add user
17     16  useradd -m -d /home/$GROUP_DIR/$username -g $GROUP_NAME -s /
    bin/bash $username
18     17
19     18  # create password
20     19  password=$(pwgen -N 1)
21     20
22     21  # add password - non-interractive mode
23     22  echo "$username:$password" | chpasswd
24     23
25     24  # set quota
26     25  setquota $username $(($QUOTA * 1024)) $(($QUOTA + 10) *
    1024)) $(($QUOTA * 10)) $(($QUOTA + 10) * 10)) -a
27     26
28     27  # send e-mail
29     28  mail -s "New Account" $email <<EndOfMsg
30     29  A new account has been created for you. Username is
    $username.
31
32     30  Your password is $password. Please use passwd to change it.
33
34     31
35     32  Your qota limit is $QUOTA MB.
36
37     33
38     34  Have a nice day!
39     35  EndOfMsg
40     36
41     37  exit 0
42
43  root@anaconda:/home/razvan/uso/scripting# ./my_add_user.sh
44
45  root@anaconda:/home/razvan/uso/scripting# cat /etc/passwd | tail -1
46  newuser:x:1051:1026::/home/students/newuser:/bin/bash
```

12.4.2 Caractere speciale shell

Shell-ul are un set de caractere rezervate, caractere ce au roluri bine definite într-un script.

Caracterul blank (spațiu)

Caracterul blank este folosit pentru a separa argumentele unei comenzi de comandă și între ele. Astfel, în exemplul

```
ubuntu@ubuntu:~$ ls -l mydir
```

avem comanda **`ls`** cu două argumente: `-l` și `mydir`. Se poate întâmpla să avem un argument care conține caractere blank. Spre exemplu, avem directorul `my dir`. Pentru afișarea conținutului acestui director, nu putem folosi comanda:

```
ubuntu@ubuntu:~$ ls my dir
```

întrucât shell-ul ar considera două argumente transmise comenzii **`ls`**. Astfel, s-ar încerca afișarea conținutului directorului `my` și apoi a directorului `dir`. Pentru a preîntâmpina acest efect, caracterul blank trebuie "citat", adică trebuie folosit ca un caracter obișnuit. Pentru aceasta există trei soluții:

1. folosirea caracterului special *ghilimele* pentru "înglobarea" numelui de director ce conține blank:

```
ubuntu@ubuntu:~$ ls "my dir"
```

2. folosirea caracterului special *apostrof*, la fel cum s-a folosit ghilimele:

```
ubuntu@ubuntu:~$ ls 'my dir'
```

3. folosirea caracterului special *backslash*, care dezactivează caracterul special de după el

```
ubuntu@ubuntu:~$ ls my\ dir
```

Despre caracterele speciale *ghilimele*, *apostrof* și *backslash* se va discuta în continuare.

Caracterul $ (dolar)

Unul dintre acestea este caracterul `$` care poate fi folosit pentru expandarea unei variabile (a valorii acesteia), a unei comenzi sau expandare aritmetică.

Expandarea unei variabile se referă la determinarea valorii acesteia:

```
razvan@anaconda:~$ a=3

razvan@anaconda:~$ b=mesaj

razvan@anaconda:~$ echo $a $b
3 mesaj
```

Expandarea unei comenzi se refera la reținerea rezultatului execuției comenzii:

```
razvan@anaconda:~$ num_dirs=$(ls -l | wc -l)

razvan@anaconda:~$ echo $num_dirs
21
```

Expandare aritmetică se referă la executarea de calcule aritmetice. Fără folosirea operatorului de expandare aritmetică, operatorii aritmetici ar fi considerați caractere simple:

```
razvan@anaconda:~$ echo 2+3+4
2+3+4

razvan@anaconda:~$ echo $((2+3+4))
9
```

Mai multe informații despre utilizarea caracterului `$` și despre expandare sunt prezentate în secțiunea 12.7.

Caracterul `$` este un caracter special. Acest lucru înseamnă că nu poate fi folosit cu semnificația de caracter direct când este urmat de un alt caracter care impune o expandare. Dacă dorim afișarea șirului `$a`, nu putem folosi comanda **`echo $a`**, pentru că ar afișa valoarea variabilei `a`:

```
razvan@anaconda:~$ echo $a
3
```

Pentru aceasta trebuie folosit caracterul *backslash* sau *apostrof*.

Folosirea ghilimelelor nu conduce la folosirea semnificației de caracter pentru `$`:

```
razvan@anaconda:~$ echo "$a"
3

razvan@anaconda:~$ echo '$a'
$a

razvan@anaconda:~$ echo \$a
$a
```

Caracterul " (ghilimele)

Un alt caracter special este " (ghilimele). Acest caracter este folosit pentru a defini șiruri de caractere:

```
$ a="sir"
$ b="acest sir"
```

Caracterul ghilimele este folosit pentru pastrarea semnificației de caracter pentru caracterele speciale: *blank*, `(`, `)`, `&`, `|` etc. Excepția o constituie caracterul `$` care își păstrează semnificația în cazul expansiunii:

```
razvan@anaconda:~$ c=3

razvan@anaconda:~$ b=a

razvan@anaconda:~$ echo "c = $c; b = $b"
c = 3; b = a
```

Pentru afișarea unui caracter ghilimele putem folosi *backslash* sau *apostrof*:

```
razvan@anaconda:~$ echo "simbolul ghilimele (\")"
simbolul ghilimele (")

razvan@anaconda:~$ echo 'simbolul ghilimele (")'
simbolul ghilimele (")
```

Caracterul \ (backslash)

S-a observat însă că nu putem afișa caracterul `$`. Comanda de mai jos afișează valoarea variabilei `a`:

```
1 razvan@anaconda:~$ echo $a
2 3
```

Shell-ul interpretează caracterul `$` ca inițiator al unei expandări. Dacă dorim afișarea șirului `$a`, nu putem folosi construcția de mai sus, întrucât se va încerca expandarea variabilei `a`. Pentru a realiza acest lucru vom folosi caracterul `\`(*backslash*). Acest caracter este caracterul de citarea și are rolul de a păstra semnificația caracterului de după el. Citare înseamnă folosirea unui caracter cu semnificația literală.

Astfel, dacă dorim afișarea șirului `$a`, folosim comanda:

```
1 razvan@anaconda:~$ echo \$a
2 $a
```

Caracterul `\` poate fi folosit și pentru citarea caracterului `"`:

```
1 razvan@anaconda:~$ echo \"
2 "
```

Tot `\` este folosit pentru afișarea `\`:

```
1 razvan@anaconda:~$ echo \\
2 \
```

Caracterul ' (apostrof)

Caracterul `'` are un rol similar cu cel al caracterului `"`. Este folosit pentru descrierea de șiruri și forțează caracterele speciale să-și păstreze semnificația. Deosebirea între ' și " este că primul este mult mai puternic. Astfel, orice caractere speciale care apar între două caractere apostrof își păstrează semnificația:

```
1 razvan@anaconda:~$ echo ' " ) ; & $ \ '
2  " ) ; & $ \
```

Alte caractere speciale

Alte caractere cu rol special în shell sunt cele folosite de operatorii shell: `>`, `<`, `|`, `&`, `;`, `(`, `)` sau cele folosite pentru expresii regulate în shell: `{`, `}`, `*`, `+`, `?`, `[`, `]`. Toate aceste caractere își păstrează semnificația dacă sunt folosite între ghilimele sau apostroafe. Mai multe detalii despre caracterele folosite pentru expresii regulate în shell se regăsec în secțiunea 12.7.2.

Dacă, spre exemplu, un utilizator dorește afișarea tuturor pachetelor instalate care încep cu litera `n`, va trebui să utilizeze comanda:

```
1 ubuntu@ubuntu:~$ dpkg -l n*
```

Comanda de mai jos nu va funcționa dacă există o intrare în directorul curent care începe cu litera `n` din cauza expandării expresiei regulate de către shell, fără a fi transmis către utilitarul **`dpkg`**:

```
razvan@anaconda:~/uso/scripting$ ls
hw.bash  new

razvan@anaconda:~/uso/scripting$ dpkg -l n*
No packages found matching new.
```

Soluția este citarea caracterului *:

```
razvan@anaconda:~/uso/scripting$ dpkg -l 'n*'
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err:
uppercase=bad)
||/ Name           Version        Description
+++-==============-==============-============================================

ii  nano           2.0.2-1etch1   free Pico clone with some new features
un  nano-tiny      <none>         (no description available)
un  nas            <none>         (no description available)
un  nas-lib        <none>         (no description available)
pn  nautilus       <none>         (no description available)
pn  nautilus-cd-bu <none>         (no description available)
ii  nautilus-data  2.14.3-11      data files for nautilus
```

12.4.3 Instrucțiuni de decizie

Instrucțiunile de decizie folosite în scripturile shell sunt `if` și `case`. O instrucțiune de decizie permite testarea unei condiții și executarea unei acțiuni sau a alteia conform condiției.

if

Vom analiza instrucțiunea `if` prin următorul exemplu: dacă utilizatorul local este `root` atunci se afișează un mesaj, dacă utilizatorul local este `razvan` se afișează un alt mesaj, iar dacă utilizatorul local este altcineva, se afișează un mesaj specific:

```
razvan@anaconda:~/uso/scripting$ cat -n if_user.bash
     1  #!/bin/bash
     2
     3  user=$(whoami)
     4
     5  if test $user = "root"; then
     6          echo "Bow before me for I am root."
     7  elif test $user = "razvan"; then
     8          echo "I am no humble user."
     9  else
    10          echo "I am but a humble user."
    11  fi
12
razvan@anaconda:~/uso/scripting$ bash if_user.bash
I am no humble user.
```

Opțiunea `-n` a comenzii **`cat`** afișează și numărul liniei. Sintaxa `if` este următoarea:

```
1 if conditie1; then
2   actiune conditie1 indeplinita
3 elif conditie2; then
4   actiune conditie2 indeplinita
5 else
6   actiune conditie neindeplinita
7 fi
```

Cuvinte cheie predefinite sunt `if`, `then`, `elif`, `else`, `fi`. *conditie1*, respectiv *conditie2* sunt condiții care sunt testate. În cazul în care o conditie este îndeplinită se execută acțiunea asociată.

După cum se vede din exemplu (liniile 5 și 7), condițiile sunt exprimate cu ajutorul utilitarului test. Acesta interpretează condiția și întoarce 0 dacă este îndeplinită sau 1 dacă nu:

```
1 razvan@anaconda:~/uso/scripting$ test "alfa" = alfa
2
3 razvan@anaconda:~/uso/scripting$ echo $?
4 0
5
6 razvan@anaconda:~/uso/scripting$ test "alfa" = alf
7
8 razvan@anaconda:~/uso/scripting$ echo $?
9 1
```

Variabila `$?` este o variabilă predefinită care deține valoarea de retur a ultimei expresii.

Același efect poate fi obținut cu ajutorul operatorului de test al shell-ului `[ .. ]`:

```
1 razvan@anaconda:~/uso/scripting$ [ "alfa" = alfa ]
2
3 razvan@anaconda:~/uso/scripting$ echo $?
4 0
5
6 razvan@anaconda:~/uso/scripting$ [ "alfa" = alf ]
7
8 razvan@anaconda:~/uso/scripting$ echo $?
9 1
```

Este obligatoriu un caracter blank (spațiu) după [și înainte de] la operatorul de test de condiție din shell.

Condiția folosită `if` nu trebuie să fie neapărat o condiție de test. Se poate folosi orice construcție care poate fi evaluată la un număr. Dacă acel număr este 0 condiția este îndeplinită și se execută acțiunea corespunzătoare, altfel nu. În exemplu următor, se afișează un mesaj corespunzător doar dacă utilizatorul indicat de variabila `user` are directorul de bază în `/home` (nu este utilizator de sistem):

```
1 razvan@anaconda:~/uso/scripting$ cat -n home_user.bash
2      1  #!/bin/bash
3      2
4      3  user="mihai"
5      4
6      5  if grep "$user" /etc/passwd | grep "/home"; then
7      6          echo "User $user is not homeless."
8      7  else
9      8          echo "User $user is homeless."
```

```
10      9  fi
11
12  razvan@anaconda:~/uso/scripting$ bash home_user.bash
13  amihaiuc:x:1011:1011:Alex Mihaiuc,,,:/home/amihaiuc:/bin/bash
14  mihai:x:1014:1014:Mihai Dobrescu,,,:/home/mihai:/bin/bash
15  mihaif:x:1023:1023:Mihai Florian,,,:/home/mihaif:/bin/bash
16  User mihai is not homeless.
```

echo -n "test" Se observă că **if** testează ieșirea comenzii compuse

```
1  grep "$user" /etc/passwd | grep "/home"
```

care verifică dacă există utilizatorul `mihai` în sistem și dacă acest utilizator are directorul de bază în `/home` (linia 5). Întrucât acesta există, afișează mesajul "User mihai is not homeless.".

Totuși, comanda **grep** afișează și liniile găsite, lucru care este deranjant în perspectiva funcționalității scriptului. Soluția este redirectarea ieșirii comenzii **grep** către `/dev/null`.

```
1   razvan@anaconda:~/uso/scripting$ cat -n home_user.bash
2        1  #!/bin/bash
3        2
4        3  user="mihai"
5        4
6        5  if grep "mihai" /etc/passwd | grep "/home" > /dev/null ; then
7        6          echo "User $user is not homeless."
8        7  else
9        8          echo "User $user is homeless."
10       9  fi
11
12  razvan@anaconda:~/uso/scripting$ bash home_user.bash
13  User mihai is not homeless.
```

Condiții de test Atât `test` cât și operatorul de condiție shell `[..]` analizează o condiție și întorc 0 dacă este îndeplinită sau 1 dacă nu este îndeplinită. În exemplele prezentate de anterior condiția a fost `sir1 = sir2`. Există mai multe condiții, o parte din care sunt prezentate în tabelul de mai jos:

Mai multe condiții pot fi folosite simultan folosind operatorii `-a` și `-o` cu sintaxele respectiv `conditie1 -a conditie2` și `conditie1 -o conditie2`.

Câteva exemple de folosire a condițiilor de mai sus sunt prezentate în continuare:

```
1   razvan@anaconda:~/uso/scripting$ test -f a.txt
2
3   razvan@anaconda:~/uso/scripting$ test -s a.txt
4
5   razvan@anaconda:~/uso/scripting$ test a.txt -ot b.txt
6
7   razvan@anaconda:~/uso/scripting$ test 3 -eq 4
8
9   razvan@anaconda:~/uso/scripting$ test -z ""
10
11  razvan@anaconda:~/uso/scripting$ test -z "a"
12
13  razvan@anaconda:~/uso/scripting$ test "a" = "b"
14
```

```
15 razvan@anaconda:~/uso/scripting$ test -f a.xt -a 3 -eq 4
```

Negarea unui condiții se face cu ajutorul operatorului `!`:

```
1 razvan@anaconda:~$ if ! test 4 -gt 5; then echo "4 nu e mai mare ca 5";
  else echo "4 e mai mare ca 5"; fi
2 4 nu e mai mare ca 5
```

Operatorul `((...))` de expansiune aritmetică poate fi folosit pentru evaluarea unor condiții într-un format asemănător C:

```
1 razvan@anaconda:~/uso/scripting$ if ((1 < 4)); then echo "adevarat"; fi
2 adevarat
```

Mai multe detalii despre operatorul `((...))` se găsesc în secțiunea 12.7.1.

case

Instrucțiunea `case` permite selectarea între mai multe șabloane de șiruri de expresii regulate. În exemplul de mai jos se verifică dacă un șir începe cu o literă mică, o literă mare sau cu un număr:

```
1  razvan@anaconda:~/uso/scripting$ cat -n case_begin.bash
2       1  #!/bin/bash
3       2
4       3  var="Alfa"
5       4
6       5  case $var in
7       6          [a-z]*) echo "litera mica";;
8       7          [A-Z]*) echo "litera mare";;
9       8          [0-9]*) echo "cifra";;
10      9          *) echo "altceva";;
11     10  esac
12
13 razvan@anaconda:~/uso/scripting$ bash case_begin.bash
14 litera mare
```

Sintaxa pentru `case`, observabilă și în exemplu, este:

```
1 case valoare_variabila in
2   sablon1) actiuni;;
3   sablon2) actiuni;;
4   ...
5
6 esac
```

Cuvinte rezervate sunt `case`, `in`, `esac`. Se verifică dacă `valoare_variabila` se potrivește cu unul dintre șabloanele ulterioare. În momentul găsirii unei potriviri se execută acțiunile asociate. Acțiunile asociate șablonului se încheie cu `;;` (două simboluri "punct și virgulă").

În exemplul de mai sus, variabila `var` se inițializează la șirul "`Alfa`" (linia 3). După aceasta se verifică dacă variabila începe cu literă mică (linia 6), cu literă mare (linia 7) sau cu cifră (linia 8). Şabloanele sunt date, de obicei, de expresii regulate. Mai multe detalii despre expresii regulate se găsesc în secțiunea 12.7.2.

Caracterul `*` înseamnă orice și este folosit ca echivalent pentru cuvântul cheie `default` din C.

12.4.4 Cicluri în shell

Bash pune la dispoziția utilizatorului trei instrucțiuni de ciclare cu funcționalitate similară dar sintaxă diferită: `for`, `while` și `do/until`. Vom exemplifica sintaxa și modul de utilizare a acestor comenzi pe un exemplu simplu: calculul sumei primelor 10 numere naturale.

Instrucțiunea `for`

Scriptul shell pentru calculul sumei primelor 10 numere naturale este următorul:

```
razvan@anaconda:~/uso/scripting$ cat -n for10.bash
     1  #!/bin/bash
     2
     3  sum=0
     4  for i in 1 2 3 4 5 6 7 8 9 10; do
     5          sum=$(($sum + $i))
     6  done
     7
     8  echo "Suma este: $sum"

razvan@anaconda:~/uso/scripting$ bash for10.bash
Suma este: 55
```

După cum se vede, sintaxa `for` este

```
for var in lista; do
  actiuni
done
```

sau

```
for var in lista
do
  actiuni
done
```

Varibila `var` este folosită pentru a parcurge lista element cu element. Cuvinte rezervate sunt `for`, `in`, `do` și `done`.

Se parcurg elementele listei (linia 4) și se execută acțiunile precizate (linia 5). În cazul de față (linia 5), folosind operatorul de expansiune aritmetică `$((...))` se incrementează suma cu valoarea lui `i`. Ca orice alte facilități shell, `for` poate fi folosit direct în linia de comandă:

```
razvan@anaconda:~/uso/scripting$ for i in 1 2 3 4; do echo $i; done
1
2
3
4
```

Acest lucru se poate extinde și la scriptul de mai sus cu pierderea parțială a clarității:

```
razvan@anaconda:~/uso/scripting$ sum=0; for i in 1 2 3 4 5 6 7 8 9 10; do
 sum=$(($sum+$i)); done; echo "Suma este: $sum"
Suma este: 55
```

sau:

```
razvan@anaconda:~$ sum=0

razvan@anaconda:~$ for i in 1 2 3 4 5 6 7 8 9 10; do
> sum=$(($sum + $i))
> done

razvan@anaconda:~$ echo $sum
55
```

Lista poate conține orice fel de element, fie număr fie șir de caractere:

```
razvan@anaconda:~/uso/scripting$ for i in abc def ghi; do echo $i; done
abc
def
ghi
```

Separatorul implicit pentru listă este caracterul *blank* (spațiu). Dacă un șir conține *blank*, acesta trebuie citat:

```
razvan@anaconda:~/uso/scripting$ for i in "abc def" ghi; do echo $i; done
abc def
ghi

razvan@anaconda:~/uso/scripting$ for i in 'abc def' ghi; do echo $i; done
abc def
ghi

razvan@anaconda:~/uso/scripting$ for i in abc\ def ghi; do echo $i; done
abc def
ghi
```

Comanda `seq` Revenind la calculul sumei primelor 10 numere naturale se observă lipsa de scalabilitate a specificării manuale a elementelor listei. Acest aspect poate fi automatizat prin folosirea comenzii **`seq`**. Aceasta permite generarea unei liste de numere într-un interval dat:

```
razvan@anaconda:~/uso/scripting$ seq 1 4
1
2
3
4

razvan@anaconda:~/uso/scripting$ seq 1 2 10
1
3
5
7
9
```

Cu două argumente, `seq` afișează elementele cuprinse între cele două limite. Cu trei argumente, argumentul din mijloc (al doilea) este folosit ca pas de incrementare.

Dacă, spre exemplu, am dori calculul sumei primelor 100 de numere naturale am folosi urmatorul script shell:

```
razvan@anaconda:~/uso/scripting$ cat -n for10_seq.bash
```

```
     1  #!/bin/bash
     2
     3  sum=0
     4  for i in $(seq 1 100); do
     5          sum=$(($sum + $i))
     6  done
     7
     8  echo "Suma este: $sum"

razvan@anaconda:~/uso/scripting$ bash for10_seq.bash
Suma este: 5050
```

O altă soluție este folosirea expandării aritmetice ca în exemplul de mai jos:

```
razvan@anaconda:~/uso/scripting$ cat -n for10_exp.bash
     1  #!/bin/bash
     2
     3  sum=0
     4  for ((i = 1; i <= 100; i++)); do
     5          sum=$(($sum + $i))
     6  done
     7
     8  echo "Suma este: $sum"

razvan@anaconda:~/uso/scripting$ bash for10_exp.bash
Suma este: 5050
```

Se observă similaritatea cu sintaxa C pentru instrucțiunea `for`.

Lucrul cu sistemul de fișiere Folosind `for` putem parcurge intrările dintr-un director al sistemului de fișier. O emulare a comenzii `ls` este următorul script shell:

```
razvan@anaconda:~/uso/scripting$ cat -n for_ls.bash
     1  #!/bin/bash
     2
     3  n=0
     4  for i in *; do
     5          n=$(($n + 1))
     6          if test -f "$i"; then
     7                  echo "f - $i"
     8          elif test -d "$i"; then
     9                  echo "d - $i"
    10          else
    11                  echo "u - $i"
    12          fi
    13  done
    14 echo "total $n"

razvan@anaconda:~/uso/scripting$ bash for_ls.bash
f - do_until10.bash
f - echo.bash
f - for10.bash
f - for10_exp.bash
f - for10_seq.bash
f - for_ls.bash
f - here.bash
f - hw.bash
d - test_dir
f - while10.bash
```

```
28 total 10
```

Folosirea caracterului `*` (linia 4) înseamnă expandarea conținutului directorului curent într-o listă de intrări care este parcursă cu ajutorul variabilei `i`. Variabila `n` (linia 3) este folosită pentru contorizarea numărului de intrări (linia 5) folosind expandare aritmetică. Se recomandă folosirea ghilimelelor în lucrul cu fișierele (linia 6) pentru a preveni apariția caracterelor cu rol special în shell (vezi secțiunea 12.4.2).

Instrucțiunea while

Calculul sumei primelor 10 numere naturale folosind `while` se realizează cu scriptul de mai jos:

```
1  razvan@anaconda:~/uso/scripting$ cat -n while10.bash
2       1  #!/bin/bash
3       2
4       3  i=1
5       4  n=10
6       5  sum=0
7       6
8       7  while test $i -le $n; do
9       8          sum=$(($sum + $i))
10      9          ((i++))
11     10  done
12     11
13     12  echo "Suma este: $sum"
14
15 razvan@anaconda:~/uso/scripting$ bash while10.bash
16 Suma este: 55
```

Sintaxa `while` este

```
1 while conditie; do
2   actiuni
3 done
```

`conditie` urmează aceleași reguli ca în cazul `if`. Exemplul prezentat seamănă foarte mult cu un program C: inițializări (liniile 2, 3, 4), condiție (linia 7), acțiuni (linia 8), incrementare (linia 9). Se poate observa că incrementarea lui `i` (linia 9) s-a realizat prin intermediul operatorului de expandare aritmetică.

Comanda `while` este folosită cu precădere în momentul în care condiția este dată de valoarea de retur a unei alte comenzi (de multe ori `read`). O emulare a comenzii **`cat`** cu opțiunea `-n` este exemplificată în scriptul următor:

```
1  razvan@anaconda:~/uso/scripting$ cat -n while_cat.bash
2       1  #!/bin/bash
3       2
4       3  i=1
5       4  cat for10.bash | while read a; do
6       5          echo -e "$i:\t$a"
7       6          i=$(($i + 1))
8       7  done
9       8
10
11 razvan@anaconda:~/uso/scripting$ bash while_cat.bash
12 1:       #!/bin/bash
```

```
2:
3:      sum=0
4:      for i in 1 2 3 4 5 6 7 8 9 10; do
5:      sum=$(($sum + $i))
6:      done
7:
8:      echo "Suma este: $sum"
```

Se observă că se citește în variabila `a` câte o linie din fișiereul `for10.bash` până la sfârșitul fișierului.

Instrucțiunea until

Comanda `until` este asemănătoare cu `while`, doar că în acest caz condiția nu este cea de continuare a acțiunilor, ci este cea de oprire. Datorită similarității nu vom insista; scriptul shell care calculează suma primelor 10 numere naturale este prezentat în continuare:

```
razvan@anaconda:~/uso/scripting$ cat -n until10.bash
     1  #!/bin/bash
     2
     3  i=1
     4  sum=0
     5  n=10
     6
     7  until test $i -gt $n; do
     8          sum=$(($sum + $i))
     9          i=$(($i + 1))
    10  done
    11
    12  echo "Suma este: $sum"

razvan@anaconda:~/uso/scripting$ bash until10.bash
Suma este: 55
```

Exemplele prezentate în secțiunea 12.4.2 sunt pur academice. Nu există niciun motiv întemeiat pentru a calcula suma numerelor naturale sau a emula comenzi existente folosind un script shell. Rolul unui script shell este acela de a folosi cele mai potrivite utilitare existente pentru a rezolva o problemă.

12.5 Filtre de text

Utilitarele de filtrare a textului primesc la intrare un fișier text și oferă la ieșire o formă prelucrată a acestuia. Prelucrarea poate impune schimbarea ordinii elementelor, selecția anumitor elemente (linii, coloane), substituția unor elemente (linii, cuvinte, caractere), adăugarea de noi elemente (număr de linie etc.) și alte operații. Aceste utilitare sunt, de obicei, folosite în tandem cu instrucțiuni de decizie sau de cicluri shell pentru obținerea unor informații ce pot fi utilizate ulterior.

12.5.1 cat, tac, nl

Utilitarul principal în prelucrarea de șiruri este **cat**. **cat** este folosit pentru afișarea unui fișier și (de cele mai multe ori) este prima comandă dintr-o înlănțuire de comenzi care folosesc `|`. După cum s-a precizat în secțiunile anterioare, o opțiune utilă a **cat** este `-n` care afișează și numărul liniei:

```
razvan@anaconda:~/uso/scripting$ cat -n nume.txt
     1  sorin
     2  stefania
     3  mihaela
     4  florin
     5  codrin
     6  lucian
     7  razvan
     8  tavi
```

Utilitarul **tac** (**cat** inversat) este folosit pentru afișarea unui fișier cu liniile inversate (prima linie ultima):

```
razvan@anaconda:~/uso/scripting$ tac nume.txt
tavi
razvan
lucian
codrin
florin
mihaela
stefania
sorin
```

Utilitarul **nl** este folosit pentru a afișa liniile împreună cu numărul lor:

```
razvan@anaconda:~/uso/scripting$ nl for10_exp.bash
     1  #!/bin/bash

     2  sum=0
     3  for ((i = 1; i <= 100; i++)); do
     4          sum=$(($sum + $i))
     5  done

     6  echo "Suma este: $sum"
```

Deosebirea față de **cat -n** este faptul că **nl** nu consideră liniile goale. De asemenea, **nl** are opțiuni pentru paginare, considerare de secțiuni etc. care depășesc aria de cuprindere a acestei cărți.

12.5.2 sort, uniq

Comanda **sort** este folosită pentru sortarea liniilor primite la intrare. În exemplul de mai jos, se realizează sortarea liniilor din fișierul `nume.txt`:

```
razvan@anaconda:~/uso/scripting$ cat nume.txt
sorin
stefania
mihaela
florin
```

```
codrin
lucian
razvan
tavi

razvan@anaconda:~/uso/scripting$ sort nume.txt
codrin
florin
lucian
mihaela
razvan
sorin
stefania
tavi
```

Se poate întâmpla ca un nume să apară de două ori și să avem nevoie doar de o singură apariție în ieșire. În această situație trebuie folosit utilitarul **`uniq`** sau opțiunea `-u` la sort:

```
razvan@anaconda:~/uso/scripting$ sort nume2.txt
alex
alina
alina
cristi
cristi
cristi
razvan
razvan

razvan@anaconda:~/uso/scripting$ sort nume2.txt | uniq
alex
alina
cristi
razvan

razvan@anaconda:~/uso/scripting$ sort -u nume2.txt
alex
alina
cristi
razvan
```

Un exemplu util a fost prezentat în secțiunea 12.4.2. În acel exemplu se dorea aflarea numărului de utilizatori care s-au autentificat în sistem într-o zi dată. Comanda utilizată a fost:

```
razvan@anaconda:~/uso/scripting$ last -30 | grep Mon | cut -d' ' -f1 |
sort | uniq | wc -l
9
```

Opțiuni utile `sort` Dintre opțiunile utile ale comenzii sort amintim:

- `-u`: elimină duplicatele după sortare;
- `-r`: sortare inversă (în ordine descrescătoare);
- `-t`: specificarea separatorului folosit pentru sortare; implicit, sortarea se realizează după primul câmp;
- `-n`: sortare numerică; implicit sortarea este alfanumerică, după cum se poate vedea în exemplul de mai jos:

```
razvan@anaconda:~/uso/scripting$ sort numere.txt
-3
10
100
12
15
2101
28
492
5

razvan@anaconda:~/uso/scripting$ sort -n numere.txt
-3
5
10
12
15
28
100
492
2101
```

Sortare avansată Uneori poate apărea situația în care dorim sortarea după o coloană a fișierului de intrare. Presupunem că avem o miniagendă cu persoane în care intrările sunt în forma `prenume, nume, număr de telefon, adresă de e-mail` și dorim sortarea după nume (adică după a doua coloană). Pentru aceasta vom folosi comanda:

```
razvan@anaconda:~/uso/scripting$ cat nume3.txt
Adrian,Munteanu,0711 111 111,am@ex.com
Ilinca,Zafiu,0711 111 112,iz@ex.com
Andreea,Popa,0711 111 113,ap@ex.com
Dan,Badea,0711 111 114,db@ex.com

razvan@anaconda:~/uso/scripting$ sort -t , -k 2,2 nume3.txt
Dan,Badea,0711 111 114,db@ex.com
Adrian,Munteanu,0711 111 111,am@ex.com
Andreea,Popa,0711 111 113,ap@ex.com
Ilinca,Zafiu,0711 111 112,iz@ex.com
```

Am folosit argumentul `-t` pentru a specifica virgula ca separator. Pentru sortarea după o cheie specifică se folosește argumentul `-k`. Acesta specifică de la ce câmp se începe sortarea (aici este vorba de al doilea câmp) și cu ce câmp se termină (în cazul de față tot 2: `-k 2,2`). În caz de egalitate, se poate alege un nou câmp de sortare prin specificarea unui nou argument `-k`.

12.5.3 head, tail

Comenzile **`head`** și **`tail`** sunt utile pentru a reține primele sau ultimele linii dintr-un fișier primit la intrare. Implicit se reține primele sau ultimele 10 linii. Exemple de utilizare sunt prezentate în continuare:

```
razvan@anaconda:~/uso/scripting$ head /etc/passwd
```

```
razvan@anaconda:~/uso/scripting$ tail /etc/passwd

razvan@anaconda:~/uso/scripting$ head -n 20 /etc/passwd

razvan@anaconda:~/uso/scripting$ tail -n 20 /etc/passwd
```

Comenzile specificate afișează respectiv: primele 10 linii din fișierul `/etc/passwd`, ultimele 10 linii, primele 20 de linii, ultimele 20 de linii.

Comanda **`tail`** este utilă pentru inspecția jurnalelor (log-urilor). Jurnalele sunt de obicei fișiere de mari dimensiuni și, pentru acestea, de cele mai multe ori este suficientă vizualizarea ultimelor intrări.

O opțiune utilă a comenzii **`tail`** este `-f`. Aceasta permite vizualizarea conținutului unui fișier în timp real în sensul că așteaptă scrierea unor noi linii în fișier și afișarea acestora pe măsură ce acestea sunt adăugate:

```
root@anaconda:~# tail -f /var/log/apache/access.log
```

Întreruperea rulării comenzii se încheie prin apăsarea `CTRL-C` (transmiterea semnalului `SIGINT`).

12.5.4 cut

Comanda **`cut`** este utilă pentru a extrage coloane dintr-un fișier format corespunzător. Spre exemplu, dacă folosim fișierul prezentat în secțiunea anterioară și vrem afișarea prenumelui și a adresei de e-mail vom folosi comanda:

```
razvan@anaconda:~/uso/scripting$ cut -d ',' -f 1,4 < nume3.txt
Adrian,am@ex.com
Ilinca,iz@ex.com
Andreea,ap@ex.com
Dan,db@ex.com
```

Cele două opțiuni cele mai importante ale comenzii cut sunt **`-d`** pentru a specifica delimitatorul și **`-f`** pentru a preciza câmpurile (coloanele) care se doresc extrase. În exemplul de mai sus, separatorul a fost virgulă și s-au reținut câmpurile (coloanele) 1 și 4 corespunzătoare prenumelui și adresei de e-mail.

Dacă se dorește extragerea numelui de utilizator și al directorului de bază pentru utilizatorii care au directorul de bază în `/home`, vom folosi următoarea comandă:

```
razvan@anaconda:~/uso/scripting$ cat /etc/passwd | grep /home | cut -d
':' -f1,6
ftp:/home/ftp
george:/home/students/george
andreic:/home/students/andreic
iepurasu:/home/students/iepurasu
valentin:/home/students/valentin
dhartescu:/home/students/dhartescu
ciconaru:/home/students/ciconaru
```

Comanda de mai sus extrage utilizatorii care au directorul de bază în `/home` (**`cat /etc/passwd | grep /home`**), după care extrage numele de utilizator și directorul de bază asociat (**`cut -d ,:' -f 1,6`**). Separatorul de câmp în `/etc/passwd`

este `:`, iar indexurile de câmp corespunzătoare numelui de utilizator și directorului de bază sunt 1 și 6.

12.5.5 tr

Utilitarul **`tr`** (*transliterate*) este folosit pentru translatarea la nivel de caracter a informațiilor de la intrare. Acțiuni posibile care pot fi efectuate cu ajutorul comenzii **`tr`** sunt:

- translatarea caracterelor,
- eliminarea caracterelor care se repetă (*squeeze*),
- ștergerea de caractere.

Exemple de translatarea caracterelor sunt:

```
1  razvan@anaconda:~/uso/scripting$ tr a b <<<"acadaeaf"
2  bcbdbebf
3
4  razvan@anaconda:~/uso/scripting$ tr acd xyz <<<"acadaeaf"
5  xyxzxexf
6
7  razvan@anaconda:~/uso/scripting$ tr a-z A-Z <<<"acadaeaf"
8  ACADAEAF
9
10 razvan@anaconda:~/uso/scripting$ tr '[:lower:]' '[:upper:]' <<<"acadaeaf"
11 ACADAEAF
```

În primul exemplu se translatează caracterul `a` în caracterul `b`. Şirul este transmis sub formă de *here string*. În cel de-al doilea exemplu, avem un set de intrare (`acd`) și un set de ieșire (`xyz`). Translatarea se face între primul element din setul de intrare și primul element din setul de ieșire, al doilea element din setul de intrare și al doilea element din setul de ieșire etc. Al treilea și al patrulea exemplu translatează literele mari în litere mici. `a-z` simbolizează literele de la `a` la `z`. De asemenea, expresiile `[:lower:]` și `[:upper:]` reprezintă, respectiv, literele mici și literele mari din alfabet. Utilitarul **`tr`** folosește mai multe astfel de expresii. Mai multe detalii puteți afla din pagina de manual sau info (**`man tr`**, **`info coreutils`**).

Exemple de eliminare a caracterelor care se repetă sunt următoarele:

```
1  razvan@anaconda:~/uso/scripting$ tr -s ' ' <<< "a    b    c"
2  a b c
3
4  razvan@anaconda:~/uso/scripting$ tr -s a-z A-Z <<< "aa    bb    cc"
5  A    B    C
```

În cazul primului exemplu, caracterul spațiu este eliminat până la o singură apariție. Opțiunea `-s` este folosită pentru a specifica acțiunea de eliminare a duplicatelor unui caracter (*squeeze*).

În cel de-al doilea exemplu se folosește simultan operațiunea de translatare și de eliminare de duplicate pe șirul de intrare "aa bb cc". Rezultatul este transformarea literelor mici în litere mari simultan cu eliminarea duplicatelor acestora.

Ştergerea caracterelor se face cu ajutorul opțiunii `-d`:

```
razvan@anaconda:~/uso/scripting$ tr -d 'a' <<< 'abcdcba'
bcdcb

razvan@anaconda:~/uso/scripting$ tr -ds 'a' 'bcd' <<< 'aabbccddccbbaa'
bcdcb
```

Opțiunea `-d` primește un set de caractere care sunt eliminate. Opțiunea `-d` poate fi completată de opțiunea `-s`. În cel de-al doilea exemplu se șterge caracterul `a` din șirul de intrare iar celorlalte caractere le sunt eliminate duplicatele.

12.5.6 wc

Utilitarul **`wc`** (*word count*) permite contorizarea numărului de linii, de caractere sau de cuvinte dintr-un fișier text. La o folosire fără opțiuni, **`wc`** afișează atât numărul de linii cât și numărul de cuvinte și caractere:

```
razvan@anaconda:~/uso/scripting$ wc < for10.bash
  8  24 102
```

Pentru afișarea numai a numărului de linii, de cuvinte sau de caractere, se folosesc, respectiv, opțiunile `-l`, `-w`, `-c`:

```
razvan@anaconda:~/uso/scripting$ wc -l < for10.bash
8

razvan@anaconda:~/uso/scripting$ wc -w < for10.bash
24

razvan@anaconda:~/uso/scripting$ wc -c < for10.bash
102
```

12.5.7 grep

Utilitarul **`grep`** și variantele sale (**`egrep`**, **`fgrep`**) sunt utilitarele de bază în prelucrarea fișierelor text. Acest utilitar permite selectarea anumitor linii dintr-un fișier text pe baza unei *expresii regulate* transmise ca argument. **`grep`** primește ca parametru șirul de căutare și, eventual, fișierul în care se face căutarea. Dacă dorim să vedem informații despre autentificarea utilizatorului `adrian` în sistem vom folosi comanda:

```
razvan@anaconda:~/uso/scripting$ last | grep adrian
adrian   pts/2        92.80.182.158    Sun Sep  2 22:30 - 22:38  (00:07)
adrian   pts/2        92.80.149.10     Sun Sep  2 09:10 - 09:29  (00:18)
adrian   pts/0        92.80.149.10     Sun Sep  2 09:06 - 09:29  (00:23)
```

Dacă dorim să afișăm informații despre utilizatorii din sistem al căror nume începe cu litera `a` vom folosi comanda:

```
razvan@anaconda:~/uso/scripting$ cat /etc/passwd | grep '^a'
alexpoke:x:1003:1003:Poke Alexandu,,,,A & C:/home/alexpoke:/bin/bash
adrian:x:1009:1009:Adrian Nistor,,,:/home/adrian:/bin/bash
amihaiuc:x:1011:1011:Alex Mihaiuc,,,:/home/amihaiuc:/bin/bash
alina:x:1016:1016:Alina Deaconescu,,,:/home/alina:/bin/bash
andrewbwm:x:1031:1026:Andrei Buhaiu:/home/students/andrewbwm:/bin/bash
```

```
alexef:x:1032:1026:Alex Eftimie:/home/students/alexef:/bin/bash
andreic:x:1045:1026:Andrei Cibotaru:/home/students/andreic:/bin/bash
```

În cazul de față, pentru obținerea utilizatorilor al căror nume începe cu litera `a` se folosește expresia regulată `^a`. Utilizarea ei va duce la selectarea acelor linii care încep cu litera `a` (caracterul `^` este special și simbolizează început de linie). Mai multe detalii despre *expresii regulate* **`grep`** se găsesc mai jos:

`grep` este utilizabil cu foarte multe opțiuni. O parte din cele mai importante sunt exemplificate în continuare pentru fișierul `grep_test.txt`:

```
razvan@anaconda:~/uso/scripting$ cat grep_test.txt
alfa beta
AlFa BeTa
Beta gamma
beta delta
deltadd epsilon
```

- opțiunea `-i` ignoră literele mari (*ignore case*):

```
razvan@anaconda:~/uso/scripting$ grep -i "alfa" < grep_test.txt
alfa beta
AlFa BeTa
```

- opțiunea `-v` inversează căutarea (afișează liniile care nu conțin expresia de căutare):

```
razvan@anaconda:~/uso/scripting$ grep -v "alfa" < grep_test.txt
AlFa BeTa
Beta gamma
beta delta
deltadd epsilon

razvan@anaconda:~/uso/scripting$ grep -v -i "alfa" < grep_test.txt
Beta gamma
beta delta
deltadd epsilon
```

- opțiunea `-n` afișează numărul liniei:

```
razvan@anaconda:~/uso/scripting$ grep -n "alfa" < grep_test.txt
1:alfa beta
```

- opțiunea `-w` este folosită pentru căutarea unui cuvânt întreg:

```
razvan@anaconda:~/uso/scripting$ grep -w "delta" < grep_test.txt
beta delta
```

- opțiunea `-r` face căutare recursivă în structura de directoare:

```
razvan@anaconda:~/uso/scripting$ grep -r "for" .

./for_ls.bash:for i in *; do
./while_cat.bash:cat for10.bash | while read a; do
./for10.bash:for i in 1 2 3 4 5 6 7 8 9 10; do
./for10_seq.bash:for i in $(seq 1 100); do
./for10_exp.bash:for ((i = 1; i <= 100; i++)); do
./if_user.bash: echo "Bow before me for I am root."
```

Rezultatul dat de utilizarea acestei opțiuni prezintă fișierele în care s-a găsit acel șir/expresie regulată și linia completă/liniile complete din fișiere.

Expresii regulate grep

După cum s-a observat, **`grep`** folosește un șir de căutare care poate avea forma unei expresii regulate.

> O **expresie regulată** este un șir de caractere utilizat pentru a se potrivi cu un alte șiruri de caractere conform unor reguli de sintaxa bine precizate. O expresie regulată conține caractere simple și caractere cu semnificație specială.

O parte din caracterele/grupările de caractere cu semnificație specială sunt prezentate în continuare:

- Caracterele `[]` sunt folosite pentru a defini *un set de caractere* de potrivit; exemple:
  - `[abcd]` înseamnă potrivire cu oricare din caracterele `a`, `b`, `c`, `d`;
  - `[a-z]` înseamnă potrivire cu oricare minusculă;
  - `[0-9a-zA-Z]` înseamnă potrivire cu orice caracter alfanumeric;
  - `[^A-Z]` înseamnă potrivire cu orice caracter mai puțin majusculele.
- Caracterul `.` înseamnă orice caracter;
- Caracterul `^` poziționat în afara unui set de caractere înseamnă potrivire cu începutul liniei;
- Caracterul `$` înseamnă potrivire cu sfârșitul liniei;
- Simbolurile `\<`, respectiv `\>` sunt folosite pentru a se potrivi cu șirul vid de la început și sfârșit de cuvânt;
- Simbolul `?` înseamnă potrivire de cel mult o dată a caracterului/grupului de caractere anterioare;
- Simbolul `*` înseamnă potrivire de zero sau mai multe ori a caracterului/grupului de caractere anterioare;
- Simbolul `+` înseamnă potrivire de o dată sau mai multe ori a caracterului/grupului de caractere anterioare;
- `{n}` înseamnă potrivire de `n` ori;
- `{n,}` înseamnă potrivire de cel puțin `n` ori;
- `{n,m}` înseamnă potrivire de cel puțin `n` ori și cel mult `m` ori.

Caracterele/grupurile de caractere prezentate mai sus pot fi concatenate pentru generarea unei expresii regulate complexe.

Informații complete despre comenzile de filtrare de text folosite în shell scripting găsiți în paginile info asociate (**`info coreutils`**).

12.5.8 sed

Utilitarul **sed** (stream editor) este folosit pentru a aplica transformări textuale unui flux de date. În mod implicit, citește de la intrare o linie, aplică operația specificată, după care afișează linia modificată.

sed poate fi considerat un editor non-interactiv. Acesta aplică un set de operații fiecărei linii din fișierul de la intrare. Întrucât în memorie se reține o singură linie, prelucrarea se poate face pe un fișier cu un număr arbitrar de linii.

Pentru a exemplifica modul de utilizare a **sed** vom folosi următorul fișier de lucru:

```
1 razvan@anaconda:~/uso/scripting$ cat sed_ex.txt
2 Andreea Popescu
3 Alin Ionescu
4 Mihai Francu
5 Calin Antonescu
6 Silvia Asavei
7 Doina Ignat
```

Sintaxa de rulare

Un exemplu obișnuit de rulare a **sed** este înlocuirea unui text cu un altul:

```
1 razvan@anaconda:~/uso/scripting$ sed 's/Alin/ALIN/' sed_ex.txt
2 Andreea Popescu
3 ALIN Ionescu
4 [...]
```

După cum se observă, sintaxa de rulare **sed** este:

```
1 sed comenzi fisier_intrare
```

În cazul în care fișierul de intrare lipsește, se folosește intrare standard. De cele mai multe ori, comenzile sunt date între caractere apostrof pentru a evita folosirea caracterelor speciale shell.

În mod implicit, **sed** afișează liniile de la intrare la ieșire. Pentru a dezactiva acest lucru se folosește opțiunea `-n`.

Sintaxa comenzi sed

După cum s-a putut observa și la exemplele anterioare, o comandă **sed** are sintaxa `[adresă][functie][argumente]`.

Adresa definește spațiul din fișier la care se aplică funcția. *Adresa* poate fi o linie a fișierului, un spațiu de linii sau o expresie ce identifică linia. În acest fel:

- `1 s/a/A/` – înlocuiește litera `a` cu `A` în prima linie a fișierului;
- `1,10 s/a/A/` – înlocuiește litera `a` cu `A` în primele zece linii ale fișierului;
- `100,$ s/a/A/` – înlocuiește litera `a` cu `A` de la linia 100 până la sfârșitul fișierului;
- `/alfa/ s/a/A/` – înlocuiește litera `a` cu `A` numai în liniile în care apare șirul `alfa`.

Funcții sed

Funcțiile cele mai utile **sed** sunt prezentate în continuare.

Funcția de substituție (s) a lui **sed** este `s` și a fost folosită în exemplele anterioare. Aceasta folosește ca argumente șirul de înlocuit și cel înlocuitor. Separatorul folosit implicit este `/` (*slash*), dar se poate folosi oricare altul:

```
s/sir_de_inlocuit/sir_inlocuitor/optiuni
```

Opțiuni utile ale funcției de substituție sunt:

- `g` (*global*) – înlocuiește toate aparițiile șirului de înlocuit
- `p` (*print*) – afișează linia la ieșire; folosit, de obicei, în conjuncție cu opțiunea `-n` pentru **sed**
- `w` (*write*) – scrie linia într-un fișier primit ca parametru

Funcția quit (q) oprește citirea de la intrare a fișierului. Exemple de folosire sunt afișarea primelor `N` linii din fișierul de la intrare:

```
razvan@anaconda:~/uso/scripting$ sed '2q' sed_ex.txt
Andreea Popescu
Alin Ionescu
```

sau afișarea primelor linii până la găsirea șirului `Calin`:

```
razvan@anaconda:~/uso/scripting$ sed '/Calin/q' sed_ex.txt
Andreea Popescu
Alin Ionescu
Mihai Francu
Calin Antonescu
```

Funcțiile print și delete (p și d) afișează, respectiv șterg linia de la intrare la ieșire.

Expresii regulate în sed

Funcțiile **sed** pot avea argumente. Întrucât **sed** lucrează pe intrare în format text, argumentele pot fi exprimate ca expresii regulate. Acestea sunt similare cu expresiile regulate grep (vezi secțiunea 12.5.7). Mai multe detalii despre acestea puteți găsi în pagina info **sed**, folosind comanda:

```
info sed "Regular Expressions"
```

Exemple

Mai jos se pot regăsi câteva exemple de utilizare a comenzii **sed**:

- translatarea unui fișier în format DOS (CR/LF) în format Unix:

```
sed 's/.$//'
```

Se observă că se elimină ultimul caracter de la sfârșitul liniei (în cazul de față caracterul `\r` – *Carriage Return*)

- translatarea unui fișier în format Unix în format DOS:

```
1 sed 's/$/\r/'
```

  Se adaugă la sfârșitul liniei caracterul `\r`.

- afișarea tuturor liniilor mai puțin ultimele 100:

```
1 sed '1,2800d'
```

- ștergerea spațiilor albe de la începutul liniilor unui fișier:

```
1 sed 's/[ \t]*//'
```

- ștergerea spațiilor albe de la sfârșitul liniilor de fișier:

```
1 sed 's/[ \t]*$//'
```

- ștergerea liniilor goale:

```
1 sed 's/^[ \t]*$//'
```

- afișarea liniilor care conțin un șir de căutare (emulare **grep**):

```
1 sed -n '/regex/p'
```

- afișarea într-un fișier a liniilor care conțin un anumit șir:

```
1 sed '/regex/w out.txt'
```

- substituția de coloane; în fișierul de exemplu vrem să înlocuim numele cu prenumele:

```
1 razvan@anaconda:~/uso/scripting$ sed 's/\([a-zA-Z]*\) \([a-zA-Z]*\)
  /\2 \1/' sed_ex.txt
2 Popescu Andreea
3 Ionescu Alin
4 Francu Mihai
5 Antonescu Calin
6 Asavei Silvia
7 Ignat Doina
```

  În exemplul de mai sus se folosește construcția `\(regex\)` pentru a stoca valoarea expresiei regulate. Aceste valori sunt folosite, respectiv, cu ajutorul construcțiilor `\1`, `\2`, `\3` etc. În situația de mai sus, numele este reținut în construcția `\2` iar prenumele în `\1`. Afișarea se face în formatul nume prenume.

12.5.9 awk

awk este un utilitar și un limbaj de programare folosit pentru prelucrarea textului pe sisteme Unix. Există mai multe versiuni de awk, versiunea cea mai întâlnită pe sisteme Linux fiind implementarea GNU, **gawk**[1]. Limbajul awk are o sintaxă asemănătoare cu limbajul C care permite operații avasate pentru formatarea fișierului text. Denumirea awk provine de la inițialele celor care au proiectat limbajul: Alfred V. Aho, Peter J. Weinberger și Brian W. Kernighan.

[1] http://www.gnu.org/software/gawk/

În forma sa cea mai simplă, awk poate fi folosit ca o versiune avansată a utilitarului **cut**. Dacă, spre exemplu, se dorește doar extragerea numelui de utilizator și a directorului home folosind fișierul `/etc/passwd` următoarele două comenzi sunt echivalente:

```
razvan@valhalla:~/carte-uso.git$ cut -d ':' -f 1,6 /etc/passwd

razvan@valhalla:~/carte-uso.git$ awk -F ':' '{ print $1,":",$6;}' /etc/
passwd
```

Comenzile de mai sus reține primul și al șaselea câmp al fiecărei linii din fișierul `/etc/passwd` folosind separatorul `:`. Opțiunea `-F` a awk precizează separatorul folosit. Argumentele `$1`, respectiv `$6` sunt folosite pentru extrage primul și al șaselea câmp.

Un avantaj important al **awk** față de **cut** este posibilitatea formatării ieșirii. În vreme ce **cut** selectează anumite coloane, **awk** permite și prelucrarea ieșirii selectate. Astfel, dacă dorim ca informații afișate să fie încadrate de caracterul `|` (*pipe*), se va folosi următoarea comandă:

```
razvan@valhalla:~/carte-uso.git$ awk -F ':' '{ printf "|%-15s|%-25s|\n",
$1, $6;}' /etc/passwd
|root           |/root                     |
|daemon         |/usr/sbin                 |
|bin            |/bin                      |
|sys            |/dev                      |
|sync           |/bin                      |
|games          |/usr/games                |
|man            |/var/cache/man            |
[...]
```

Instrucțiunea `printf` folosită în exemplul de mai sus are o sintaxă similară celei din C, cu absența parantezelor.

Sintaxa awk este similară sintaxei sed:

```
awk [options] ' program instructions ' [input file]
```

În general, programul awk este aplicat fiecărei linii citită de la intrare. awk permite folosirea clauzelor speciale `BEGIN` și `END` care permit aplicarea instrucțiunilor asociate doar la începutul și sfârșitul prelucrării. Astfel, pentru a încadra ieșirea de mai sus, se folosește o comandă ca cea de mai jos. Programul se poate introduce pe mai multe linii, până la "închiderea" acestuia folosind caracterul `'` (apostrof):

```
razvan@valhalla:~/carte-uso.git$ awk -F ':' '
> BEGIN { printf "+---------------+-------------------------+\n"; }
>       { printf "|%-15s|%-25s|\n", $1, $6; }
> END   { printf "+---------------+-------------------------+\n"; }
> ' /etc/passwd
+---------------+-------------------------+
|root           |/root                     |
|daemon         |/usr/sbin                 |
|bin            |/bin                      |
|sys            |/dev                      |
[...]
|pulse          |/var/run/pulse            |
|saned          |/home/saned               |
+---------------+-------------------------+
```

awk folosește instrucțiuni și comenzi similare limbajului C. Dacă se dorește afișarea din 5 în 5 a utilizatorilor sistemului și a directorului home asociat se poate folosi o comandă de forma:

```
razvan@valhalla:~/school/2009-2010/uso/carte-uso.git$ awk -F ':' -f
print_users.awk /etc/passwd
+---------------+--------------------------+
|root           |/root                     |
|games          |/usr/games                |
|uucp           |/var/spool/uucp           |
|irc            |/var/run/ircd             |
|messagebus     |/var/run/dbus             |
|gdm            |/var/lib/gdm              |
|mpd            |/var/lib/mpd              |
|postfix        |/var/spool/postfix        |
|pulse          |/var/run/pulse            |
+---------------+--------------------------+
```

Opțiunea `-f` permite selectarea fișierul care să conțină programul awk, prezentat în listingul 12.5.9. Variabila `current_users` reține numărul de utilizatori curent. Este inițializată cu 0 la începutul prelucrării (în cadrul blocului `BEGIN`) și apoi este incrementată pe parcursul prelucrării liniilor. Se folosește instrucțiunea `if` pentru a selecta doar anumiți utilizatori.

```
#!/usr/bin/awk -f

BEGIN {
        current_users = 0;
        printf "+---------------+--------------------------+\n";
}
{
        current_users++;
        if (current_users % 5 == 1)
                printf "|%-15s|%-25s|\n", $1, $6;
}
END {
        printf "+---------------+--------------------------+\n";
}
```

Listing 12.1: Afișare utilizatorilor din 5 în 5

awk permite folosirea expresiilor regulate pentru selectarea doar anumitor linii. Astfel, dacă se dorește afișarea doar acelor utilizatori care au directorul home în `/home` se folosește un program ca cel prezentat în listingul 12.2. Expresiile regulate se scriu între caractere / (slash) similar utilitarului `sed`. Rezultatul rulării acestui program este:

```
razvan@valhalla:~/carte-uso.git$ awk -F ':' -f print-home-users.awk /etc/
passwd
+---------------+--------------------------+
|razvan         |/home/razvan              |
|festival       |/home/festival            |
|ntp            |/home/ntp                 |
|alina          |/home/alina               |
|ftp            |/home/ftp                 |
|saned          |/home/saned               |
+---------------+--------------------------+
```

```
#!/usr/bin/awk -f

```

```
3  BEGIN {
4          printf "+---------------+-------------------------+\n";
5  }
6  /\/home/ {
7          printf "|%-15s|%-25s|\n", $1, $6;
8  }
9  END {
10         printf "+---------------+-------------------------+\n";
11 }
```

Listing 12.2: Afișare utilizatori /home

În listingul 12.2 se verifică dacă fiecare linie conține șirul `/home`. De fapt, funcționarea corectă a programului se bazează doar pe verificarea faptului că a șasea coloană conține șirul `/home`. Acest lucru se poate realiza ușor în awk, după cum este prezentat în listingul 12.8.1. Construcția `$6 ~ /^\/home/` verifică dacă al șaselea câmp al unei linii începe cu șirul `/home`.

```
1  #!/usr/bin/awk -f
2
3  BEGIN {
4          printf "+---------------+-------------------------+\n";
5  }
6  $6 ~ /^\/home/ {
7          printf "|%-15s|%-25s|\n", $1, $6;
8  }
9  END {
10         printf "+---------------+-------------------------+\n";
11 }
```

Listing 12.3: Afișare utilizatori /home folosind expresii regulate

Un alt avantaj important al comenzii `awk` față de comanda `cut` este folosirea unei expresii regulate pe post de separator. Opțiunea `-F` a awk primește ca argument o expresie regulată care este folosită pentru a separa câmpurile unei linii. Următoarele două comenzi, pentru determinarea adresei hardware a interfeței eth0 sunt echivalente:

```
1  razvan@valhalla:~/carte-uso.git$ /sbin/ifconfig eth0 | head -1 | tr -s '
   ' | cut -d ' ' -f 5
2  00:1d:09:b4:0c:26
3
4  razvan@valhalla:~/carte-uso.git$ /sbin/ifconfig eth0 | head -1 | awk -F
   '[
5  \t]+' '{ print \$5; }'
6  00:1d:09:b4:0c:26
```

Expresia regulată `[ \t]+` specifică faptul că orice set de spații albe (caractere blank sau TAB) va fi folosit ca separator.

awk oferă un set mult mai amplu de funcționalități. Mai multe detalii despre acestea și despre cazuri posibile de utilizare găsiți în manualul Gawk[1].

[1] http://www.gnu.org/software/gawk/manual/

12.6 Comenzi de lucru cu fișiere

Comenzile de lucru cu fișiere sunt comenzi care permit căutarea în sistemul de fișiere și executarea unor acțiuni pentru fișierele căutate. Comenzile prezentate în această secțiune sunt **xargs**, **locate** și **find**.

12.6.1 xargs

Comanda **xargs** permite transmiterea de argumente unei alte comenzi, argumente ce pot fi trimise de la intrarea standard. Spre exemplu, dacă avem un fișier ce conține o listă fișiere pe care dorim să le ștergem (câte unul pe linie), vom folosi comanda: **xargs rm < file.txt** ca în exemplul de mai jos

```
1  razvan@anaconda:~/uso/scripting$ cat file.txt
2  tmp1.txt
3  tmp2.txt
4  tmp3.txt
5
6  razvan@anaconda:~/uso/scripting$ ls tmp*
7  tmp1.txt  tmp2.txt  tmp3.txt
8  razvan@anaconda:~/uso/scripting$ xargs rm < file.txt
9
10 razvan@anaconda:~/uso/scripting$ ls tmp*
11 ls: tmp*: No such file or directory
```

În exemplu, fișierul file conținea trei nume de fișier: `tmp1.txt`, `tmp2.txt`, `tmp3.txt`. Folosind **xargs** se invocă **rm**. Argumentele pentru **rm** sunt citite de la intrarea standard în care a fost redirectat fișierul `file.txt`. Rezultatul este ștergerea fișierelor `tmp1.txt`, `tmp2.txt` și `tmp3.txt` precizate în fișierul file.txt.

Dacă un utilizator dorește afișarea fișierelor ce conțin șirul `for` va utiliza comanda:

```
1  ubuntu@ubuntu:~$ grep -l 'for' *
```

sau

```
1  ubuntu@ubuntu:~$ ls | xargs grep -l 'for'
```

La fel, dacă se dorește generarea unei liste a utilizatorilor în sistem se poate utiliza comanda:

```
1  razvan@anaconda:~/uso/scripting$ cut -d ':' -f 1 < /etc/passwd | sort |
   tr -s '\n' ' '
2  Debian-exim adrian alexef alexpoke alina amihaiuc [...]
```

sau

```
1  razvan@anaconda:~/uso/scripting$ cut -d ':' -f 1 < /etc/passwd | sort |
   xargs echo
2  Debian-exim adrian alexef alexpoke alina amihaiuc [...]
```

Opțiuni utile pentru **xargs** sunt:

- `-d` este folosită pentru a specifica delimitatorul de parametri; implicit acesta este orice caracter alb (*whitespace*); în exemplul de mai jos se șterg fișierele `tmp1.txt`, `tmp2.txt` și `tmp3.txt`:

```
razvan@anaconda:~/uso/scripting$ cat file2.txt
tmp1.txt,tmp2.txt,tmp3.txt

razvan@anaconda:~/uso/scripting$ ls tmp*
tmp1.txt  tmp2.txt  tmp3.txt

razvan@anaconda:~/uso/scripting$ xargs -d ',' rm  < file2.txt
rm: cannot remove '\n': No such file or directory
```

Ultima linie apare ca urmare a faptului că se interpretează caracterul newline (`\n`) ca un argument posibil.

- `-0` impune folosirea ca delimitator de argument a nul-terminatorului de sir (`\0`); acest lucru este util în momentul în care parametrii conțin caractere speciale (blank, backslash etc.); este folosit, de obicei, împreună cu argumentul `-print0` la **`find`**, precum este descris în secțiunea 12.6.3.

- `-I` permite folosirea unui șir care va fi substituit pe parcursul comenzii cu argumentul primit la intrare; în următorul exemplu, fișierele primite la intrare sunt copiate cu extensia `.bkup` în `/tmp`:

```
razvan@anaconda:~/uso/scripting$ cat file3.txt
for10.bash
for10_seq.bash
for10_exp.bash

razvan@anaconda:~/uso/scripting$ xargs -I abc cp abc /tmp/abc.bkup <
 file3.txt

razvan@anaconda:~/uso/scripting$ ls /tmp/*.bkup
/tmp/for10.bash.bkup  /tmp/for10_exp.bash.bkup  /tmp/for10_seq.bash.
bkup
```

Se observă că s-a folosit șirul `abc` ca șir de substituit (prin intermediul opțiunii `-I`). La apariția acestui șir în comanda folosită de **`xargs`**, acesta a fost înlocuit cu parametrul de la intrare (în cazul nostru fiecare din cele trei fișiere). Astfel, comanda

```
ubuntu@ubuntu:~$ xargs -I abc cp abc /tmp/abc.bkup
```

este echivalentă cu

```
ubuntu@ubuntu:~$ cp for10.bash for10.bash.bkup

ubuntu@ubuntu:~$ cp for10.bash for10_seq.bash.bkup

ubuntu@ubuntu:~$ cp for10.bash for10_exp.bash.bkup
```

12.6.2 locate

Comanda **`locate`** permite localizarea de fișiere al căror nume corespunde unui anumit șablon transmis ca parametru. Comanda locate folosește o bază de date construită cu ajutorul comenzii **`updatedb`**.

Un exemplu este prezentat în continuare:

```
razvan@anaconda:~/uso/scripting$ time locate strace
/usr/bin/strace
/usr/share/doc/strace
/usr/share/doc/strace/changelog.Debian.gz
/usr/share/doc/strace/changelog.gz
/usr/share/doc/strace/copyright
/usr/share/doc/strace/TODO.gz
/usr/share/man/man1/strace.1.gz
/usr/share/vim/vim70/syntax/strace.vim
/var/lib/dpkg/info/strace.list

real    0m0.288s
user    0m0.284s
sys     0m0.004s
```

După cum se vede comanda locate este foarte rapidă întrucât folosește baza de date generată/actualizată anterior de **updatedb**. Întrucât este posibil ca anumite fișiere să fi fost șterse de la ultima actualizare, se poate folosi opțiunea `-e` pentru a afișa doar fișierele care există în sistem dintre cele cunoscute:

```
razvan@anaconda:~/uso/scripting$ locate -e strace
/usr/bin/strace
/usr/share/doc/strace
/usr/share/doc/strace/changelog.Debian.gz
/usr/share/doc/strace/changelog.gz
[...]
```

Comanda **locate** este de multe ori folosită împreună cu **xargs**:

```
razvan@anaconda:~/uso/scripting$ locate -e strace | xargs file
/usr/bin/strace:                                ELF 32-bit LSB executable,
Intel 80386, version 1 (SYSV), for GNU/Linux 2.4.1, dynamically linked (
uses shared libs), for GNU/Linux 2.4.1, stripped
/usr/share/doc/strace:                          directory
/usr/share/doc/strace/changelog.Debian.gz: gzip compressed data, was "
changelog.Debian", from Unix, last modified: Fri Oct 27 00:22:12 2006,
max compression
[...]
```

12.6.3 find

Comanda **find** este principala comandă utilizată pentru căutarea în sistemul local de fișiere. Comanda realizează o căutare recursivă în structura de directoare și descoperă acele intrări în sistemul de fișiere care satisfac o anumită condiție. Suplimentar, comanda **find** permite executarea unei acțiuni specifice.

Sintaxa **find** este mai complexă decât a altor comenzi și o vom prezenta după următorul exemplu:

```
razvan@anaconda:~$ find /usr/include/ -type f -name '*term*.h' -print
/usr/include/asm-generic/termios.h
/usr/include/linux/termios.h
/usr/include/asm/termbits.h
/usr/include/asm/termios.h
/usr/include/bits/termios.h
/usr/include/sys/termios.h
```

```
/usr/include/termio.h
/usr/include/termios.h
[...]
```

În exemplul de mai sus, se caută recursiv în `/usr/include` fișierele (`-type f`) al căror nume corespunde expresiei regulate `'*term*.h'` și se afișează acele fișiere (`-print`).

Astfel, sintaxa `find` este

```
find director optiuni_potrivire actiuni
```

unde:

- `director` este directorul în care se face căutarea (recursiv);
- `optiuni_potrivire` sunt opțiuni de căutare în sistemul de fișiere (adâncimea căutării, tipul intrării, nume, deținător, permisiuni etc.);
- `actiuni` reprezintă acțiunile întreprinse în momentul găsirii unei intrării corespunzătoare; acțiunea implicită este -print (de afișare).

Opțiuni de potrivire utile sunt:

- `tipul` intrării se precizează cu ajutorul opțiunii `-type`; astfel, `-type f` înseamnă un fișier, `-type d` înseamnă un director, `-type l` înseamnă o legătură simbolică etc.; exemplu: afișarea legăturilor simbolice din ierarhia utilizatorului curent:

```
razvan@anaconda:~$ find /home/razvan/public_html/ -type l
/home/razvan/public_html/pub/books
/home/razvan/public_html/pictures
/home/razvan/public_html/movies
/home/razvan/public_html/games
/home/razvan/public_html/wiki
/home/razvan/public_html/school/uso/cursuri
[...]
```

- numele intrării se precizează cu ajutorul opțiunii `-name` urmată de o expresie regulată descriind numele; opțiunea `-wholename` este folosită pentru potrivire cu numele complet al intrării (cale absolută); exemplu: afișarea directoarelor ce conțin calea `'c++'`:

```
razvan@anaconda:~$ find /usr/include/ -type d -wholename '*/c++/*'
/usr/include/c++/4.1.2
/usr/include/c++/4.1.2/backward
/usr/include/c++/4.1.2/bits
/usr/include/c++/4.1.2/debug
/usr/include/c++/4.1.2/ext
/usr/include/c++/4.1.2/ext/pb_assoc
[...]
```

- permisiunile se precizează cu ajutorul opțiunii `-perm`; astfel, `-perm 644` va găsi intrările care au drepturi de citire și scriere pentru utilizator și doar de citire pentru grup și pentru ceilalți; exemplu: afișarea executabilelor suidate de sistem:

```
razvan@anaconda:~$ find /bin -perm 4755
/bin/su
/bin/mount
```

```
4 /bin/umount
5 /bin/ping
6 /bin/ping6
```

- timpul ultimului acces sau a ultimei modificări cu ajutorul opțiunilor `-amin`, `-atime`, `-mmin`, `-mtime`; `-amin 20` înseamnă că fișierul a fost accesat în urmă cu 20 de minute, `-amin -20` înseamnă că fișierul a fost accesat în urmă cu cel mult 20 de minute, `-amin +20` înseamnă că fișierul a fost accesat în urmă cu cel puțin 20 de minute; exemplu: aflarea fișierelor modificate în urmă cu mai mult de 100 de zile:

```
1 razvan@anaconda:~$ find /home/razvan/ -type f -mtime +100
```

- spațiul ocupat pe disc cu ajutorul opțiunii `-size`; `-size +10M` va selecta fișierele care ocupă mai mult de 10 MB; exemplu: aflarea fișierelor care ocupă mai mult de 200 MB:

```
1 razvan@anaconda:~$ find /home/razvan/ -size +200M
```

- deținătorul fișierului cu ajutorul opțiunii `-user`;
- grupul fișierului cu ajutorul opțiunii `-group`.

Acțiunile ce pot fi întreprinse cu ajutorul comenzii **`find`** sunt:

- afișarea fișierului se face folosind `-print`; această acțiune este implicită;
- afișarea fișierelor folosind nul-terminatorul ca separator folosind `-print0`; opțiunea este utilă în combinație cu argumentul `-0` la **`xargs`**; exemplu: mutarea fișierelor `.iso` pe o nouă partiție:

```
1 ubuntu@ubuntu:~$ find . -type f -name *.iso -print0 | xargs -0 -I
abc mv abc /mnt/hda6
```

- ștergerea fișierului cu ajutorul `-delete`; exemplu: ștergerea fișierelor temporare din ierarhia utilizatorului local:

```
1 razvan@anaconda:~$ find . -type f -name '*~' -delete
```

- execuția unei comenzi precizate de utilizator cu ajutorul opțiunii `-exec`; șirul `{}` este utilizat pentru înlocuirea intrării găsite în comanda de după `exec`; comanda trebuie să se termine cu `;` (punct și virgulă); se recomandă ca atât șirul `{}` cât și caracterul `;` să fie citate pentru a preveni interpretarea lor ca niște caractere speciale; exemple:
  - crearea unei copii pentru fișierele mai vechi de 1 an din directorul curent:

```
1 razvan@anaconda:~/uso/scripting$ find . -type f -mtime +365 -
exec cp '{}' /tmp/'{}'.bkup ';'
```

  - adăugarea dreptului de execuție pentru toate directoarele:

```
1 razvan@anaconda:~/uso/scripting$ find . -type d -exec chmod a+x
'{}' ';'
```

  - afișarea fișierelor C în care este definită funcția `main`:

```
1 razvan@anaconda:~$ find school/2001-2006_code/ -type f -name '*.
c' -exec grep -l 'int main*' '{}' ';'
```

Acțiuni asupra fișierelor găsite cu find pot fi executate cu ajutorul opțiunii `-exec` sau prin folosirea **`xargs`**.

Exemple

În continuare sunt prezentate câteva exemple utile de lucru cu **`find`**:

- editarea fișierelor `.c` din ierarhia curentă care folosesc funcția `printf`:

```
1 razvan@anaconda:~$ vi $(find school/2001-2006_code/ -type f -name
  '*.c' -exec grep -l -w 'printf' '{}' ';')
```

- arhivarea fișierelor mai vechi de 30 de zile:

```
1 razvan@anaconda:~/uso/scripting$ find . -depth -type f -mtime 30 |
  tar --files-from=- --null -czf out.tgz
```

- ștergerea fișierelor utilizatorului local din `/tmp`:

```
1 razvan@anaconda:~/uso/scripting$ find /tmp -type f -user $(whoami) -
  delete
```


Informații complete de comenzile de lucru cu fișierele se găsesc în paginile info asociate (**`info find`**).

12.7 Expandarea în shell

De-a lungul secțiunilor anterioare s-au prezentat diferite metode de expandare în shell. Expandarea unei variabile la valoarea ei, expandarea unei comenzi, expandare aritmetică. Secțiunea de față oferă o recapitulare structurată a acestor mecanisme.

12.7.1 Simbolul $

Simbolul `$` este folosit pentru expandarea unei variabile, comenzi sau expresii și posibilitatea transmiterii rezultatului acesteia. Caracterul are o semnificație specială, drept pentru care va trebui citat (folosind ghilimele, apostrof, backslash) în cazul în care se dorește folosirea sa literală.

Variabile

O variabilă are un nume și o valoare. Așa cum s-a specificat, o variabilă nu are tip în shell. Poate fi interpretată ca număr sau ca șir în funcție de situație. Spre exemplu, în niciunul din cazurile de test de mai jos nu se generează eroare:

```
1 razvan@anaconda:~/uso/scripting$ b=3
2
3 razvan@anaconda:~/uso/scripting$ test "a" = $b
4
5 razvan@anaconda:~/uso/scripting$ test 1 -eq $b
```

Numele variabilei poate fi încadrat între acolade. Acestea sunt opțiunale dar sunt utile pentru a proteja numele variabilei de caracterele următoare. În exemplul de mai jos utilizarea acoladelor duce la afișarea dorită, altfel nu ar fi fost afișat nimic, variabila `ba` nefiind inițializată:

```
razvan@anaconda:~/uso/scripting$ echo ${b}a
3a
```

Folosind caracterul `$` se pot face diverite expandări asupra variabilei:

- determinarea numărului de litere compun variabila:

```
razvan@anaconda:~/uso/scripting$ var="alfanumeric"

razvan@anaconda:~/uso/scripting$ echo ${#var}
11
```

- porțiuni din numele variabilei:

```
razvan@anaconda:~/uso/scripting$ echo ${var:3}
anumeric

razvan@anaconda:~/uso/scripting$ echo ${var:3:6}
anumer
```

Expandarea unei comenzi

Expandarea unei comenzi înseamnă executarea acesteia și reținerea rezultatului comenzii. Acest lucru se poate face în două moduri:

```
$(comanda)
```

sau

```
`comanda`
```

(caracterul folosit este ` – *backquote* – se află pe tasta cu ~ de lânga tasta `1`) Rezultatul unei comenzi poate fi reținut într-o variabilă și folosit ulterior.

Expandarea aritmetică

Expandarea aritmetică presupune evaluarea unei expresii și reținerea rezultatului acesteia. Pentru aceasta se folosește operatorul `$((...))` sub forma `$(( expresie ))`.

Dacă nu este nevoie de reținerea rezultatului se poate folosi operatorul `((...))`

```
razvan@anaconda:~/uso/scripting$ a=1

razvan@anaconda:~/uso/scripting$ ((a++))

razvan@anaconda:~/uso/scripting$ echo $a
2
```

12.7.2 Expresii regulate în shell

Shell-ul permite interpretarea expresiilor regulate pentru a permite expandarea numelor de fișiere.

Construcțiile care permit expandarea numelor de fișiere sunt:

- acoladele `{}` permit precizarea unor opțiuni pentru expandare; astfel, expresia `{a,b,c}.d` se va expanda la `a.d b.d c.d`;
- `*` înseamnă expandarea la orice șir de caractere; astfel *.c înseamnă toate fișiere care au extensia `.c`, iar `a*.c` reprezintă toate fișierele care încep cu litera `a` și se termină în `.c`;
- `?` înseamnă potrivirea cu un singur caracter (oricare ar fi acela);
- `[ ... ]` permit specificarea unei clase de caractere; astfel, `[a-z]` înseamnă litere mici `[a-d0-3A-D]` înseamnă literele de la `a` la `d`, de la `A` la `D` sau cifrele de la `0` la `3`; `[^a-z]` înseamnă orice mai puțin literele de la `a` la `z`.

Exemple de utilizare a expresiilor regulate shell sunt prezentate în continuare:

- afișarea tuturor fișierelor C din directorul curent:

```
1 ubuntu@ubuntu:~$ ls *.c
```

- afișarea tuturor fișierelor de tip png, jpg sau gif:

```
1 ubuntu@ubuntu:~$ ls *.{png,jpg,gif}
```

- afișarea tuturor fișierelor al căror nume începe cu 3 cifre:

```
1 ubuntu@ubuntu:~$ ls [0-9][0-9][0-9]*
```

12.8 Parametrii unui script shell

La fel ca orice alt program, un script shell poate primi argumente în linia de comandă. Acestea pot fi folosite pentru a oferi un mod de configurare a scriptului în momentul execuției.

În C, modul în care se realizează accesul la argumentele în linia de comandă este prin intermediul variabilelor `argc` și `argv`. Astfel, variabila `argc` indică numărul de argumente primite în linia de comandă (incluzând numele executabilului), iar `argv` este vectorul de argumente: `argv[0]` este numele executabilului, `argv[1]` numele primului argument, `argv[argc-1]` este ultimul argument.

În Bash, accesul la argumente se realizează prin intermediul variabilei `$#`, reprezentând numărul de argumente, și al variabilelor `$1`, `$2` etc. reprezentând argumentele efective. Ca exemplu, vom folosi scriptul de mai jos, care nu face altceva decât să afișeze argumentele primite:

```
1 razvan@anaconda:~/uso/scripting$ cat -n args.bash
2      1  #!/bin/bash
3      2
4      3  echo "Scriptul are $# argumente."
```

```
5        4  echo "Numele scriptului este $0."
6        5  echo "Argumentele scriptului sunt '$@'."
7        6
8        7  i=1
9        8
10       9  while test $i -le $#; do
11      10          echo "Argumentul $i este ${!i}."
12      11          ((i++))
13      12  done
14
15  razvan@anaconda:~/uso/scripting$ bash args.bash alfa beta gamma delta
16  Scriptul are 4 argumente.
17
18  Numele scriptului este args.bash.
19
20  Argumentele scriptului sunt 'alfa beta gamma delta'.
21
22  Argumentul 1 este alfa.
23
24  Argumentul 2 este beta.
25
26  Argumentul 3 este gamma.
27
28  Argumentul 4 este delta.
```

În cazul exemplului de mai sus, se trimit 4 argumente. Dacă variabila `argc` numără inclusiv executabilul, variabila `$#` contorizează doar argumentele. Variabila `$0` conține numele executabilului. Variabila `$@` conține toate argumentele separate prin spațiu (*blank*) (linia 5). Folosind variabila `i` se parcurg toate argumentele primite în linia de comandă (linia 9) și se afișează (linia 10). Se observă că am folosit construcția `$!i`, pentru a expanda variabila `i` și apoi variabila care reprezintă argumentul primit din linia de comandă.

Un al mod de a realiza parcurgerea argumentelor primite în linia de comandă ar fi fost următorul:

```
1  i=1
2  for arg in $@; do
3          echo "Argumentul $i este $arg."
4          ((i++))
5  done
```

Variabilele `$1`, `$2`, ..., `$N`, reprezentând argumentele primite din linia de comandă, poartă numele de parametri poziționali.

12.8.1 Comanda shift

Comanda **`shift`**, după cum îi spune numele, este folosită la deplasarea argumentelor din linia de comandă în parametri poziționali. În exemplul de mai jos, se face parcurgerea parametrilor poziționali folosind comanda **`shift`**:

```
1  i=1
2  while ! test -z $1 ; do
3          echo "Argumentul $i este $1."
4          ((i++))
5          shift
```

```
6 done
```

Comanda **shift** este responsabilă cu deplasarea spre stânga a argumentelor unui script shell. Astfel, după rularea comenzii **shift**, se pierde informația despre primul argument, variabila `$#` scade cu 1, variabila `$1` reprezintă al doilea argument, variabila `$2` reprezintă al treilea argument etc.

Comanda **shift** poate primi un argument reprezentând contorul de deplasare. Implicit (fără argument) acesta este 1.

12.8.2 Parametri speciali

După cum a fost prezentat anterior, există variabile speciale folosite de shell pentru prelucrarea argumentelor în linia de comandă:

- `$#` reprezintă numărul de argumente din linia de comandă;
- `$0` reprezintă numele scriptului shell;
- `$1`, `$2`, ..., `$N` reprezintă argumentele din linia de comandă;
- `$@` reprezintă lista de argumente separate prin spații.

În afara acestor variabile mai există câțiva parametri speciali, prezentați în continuare:

- `$?` reprezintă valoarea de retur a ultimei comenzi;
- `$$` reprezintă pid-ul procesului bash curent;
- `$!` reprezintă pid-ul ultimului proces lansat în fundal.

12.8.3 Exemplu de utilizare a parametrilor

Un exemplu util este un script shell care să ușureze lucrul cu comanda **find** în momentul căutării unor fișiere după nume. Scriptul este prezentat în continuare:

```
1  #!/bin/bash
2
3  if test $# -ne 2; then
4          echo "Two␣arguments␣are␣required."
5          exit 1
6  fi
7
8  if ! test -d $1; then
9          echo "First␣argument␣is␣not␣a␣directory."
10         exit 1
11 fi
12
13 find "$1" -type f -name "$2"
```

Listing 12.4: Fișierul `myfind.bash`

```
1 razvan@anaconda:~/uso/scripting$ bash myfind.bash . "*.bash"
2 ./while10.bash
3 ./hw.bash
4 ./for_ls.bash
```

```
5 ./service_ip.bash
6 ./while_cat.bash
7 ./rss.bash
8 [...]
```

Se observă că se transmit ca argumente directorul în care se face căutarea și expresia regulată asociată numelui fișierului. În afara apelării **`find`** (linia 13), se verifică numărul de argumente (liniile 3-6) și dacă primul argument este un director (liniile 8-11).

12.9 Funcții

La fel ca majoritatea limbajelor de programare, shell scriptingul permite folosirea de funcții. În general funcțiile sunt bucăți de cod folosite pentru a separa o funcționalitate sau pentru refolosire (pentru a nu se scrie de două ori același cod). De asemenea, funcțiile au un rol important în mărirea lizibilității codului.

În shell scripting, funcțiile intervin în cazul scripturilor de dimensiune mai mare, unde există bucăți de cod care sunt folosite de mai multe ori.

12.9.1 Sintaxa unei funcții

Prezentăm sintaxa comenzii prin următorul exemplu, în care se poate șterge sau adăuga un utilizator în sistem:

```
1  #!/bin/bash
2
3  if test $# -ne 1; then
4          echo "One␣argument␣required:␣a␣or␣d␣(add␣or␣delete)."
5          exit 1
6  fi
7
8  username=testuser
9  groupname=testgroup
10
11 function add ()
12 {
13         cat /etc/group | grep $groupname &> /dev/null
14         if test $? -eq 1; then           # no group
15                 addgroup $groupname
16                 echo "Group␣$groupname␣didn't␣exist␣so␣it␣was␣created."
17         fi
18         useradd -m -d /home/$username -g $groupname -s /bin/bash
              $username
19         echo "User␣$username␣added.␣Use␣passwd␣to␣change␣password."
20 }
21
22 function delete ()
23 {
24         userdel -r $username
25         echo "User␣$username␣deleted."
26 }
27
28 case $1 in
```

```
29          "a") add;;
30          "d") delete;;
31          *) echo "Required arguments are 'a' or 'd'."; exit 1;;
32  esac
```

Listing 12.5: Fișierul `user.bash`

```
1  root@anaconda:/home/razvan/uso/scripting# bash user.bash a
2  Adding group 'testgroup' (GID 1031) ...
3
4  Done.
5
6  Group testgroup didn't exist so it was created.
7
8  User testuser added. Use passwd to change password.
9
10 root@anaconda:/home/razvan/uso/scripting# bash user.bash d
11 User testuser deleted.
```

În exemplul de mai sus funcțiile sunt `add` și `delete` și sunt folosite, respectiv, pentru a adăuga și a elimina un utilizator în/din sistem. Sintaxa unei definiții de funcție este:

```
1  function nume_functie ()
2  {
3    cod shell
4  }
```

Se observă că o funcție shell are un antet și un corp. În antet se prezintă numele funcției (`nume_functie`), iar în corp se regăsesc comenzile ce se execută în momentul apelării funcției. Cuvântul `function` este cuvânt rezervat și poate fi omis. Astfel, o funcție definită:

```
1  function testare ()
2  {
3    ...
4  }
```

poate fi definită și astfel:

```
1  testare ()
2  {
3    ...
4  }
```

Pentru a apela o funcție se folosește direct numele funcției urmat de parametri, exact ca în cazul apelului unei comenzi.

12.9.2 Parametrii unei funcții

În mod evident, o funcție poate primi și argumente. Modul de definire funcției nu se schimbă; altfel spus, într-un shell script o funcție nu are parametri formali. Totuși, modul de apel se schimbă prin precizarea parametrilor pentru funcție.

Pentru a exemplifica modul în care se transmit parametrii unei funcții vom folosi un exemplu de calcul a sumei unor numere naturale transmise ca argumente funcției `sum-func`:

```
#!/bin/bash

function sum_func ()
{
        sum=0
        while ! test -z $1; do
                sum=$(($sum + $1))
                shift
        done
}

sum_func 1 2 3 4 5
echo "Sum: $sum"

sum_func 2 3 5 7 11 13 17 19
echo "Sum: $sum"
```

Listing 12.6: Fișierul `sum-func.bash`

```
razvan@anaconda:~/uso/scripting$ bash sum-func.bash
Sum: 15
Sum: 77
```

Se observă că argumentele unei funcții se obțin tot folosind parametri poziționali (`$1`, `$2` etc.). Variabila `$#` reprezintă numărul de argumente transmise funcției. Variabila `$0` rămâne neschimbată și reprezintă numele scriptului. De asemenea, se poate folosi comanda **`shift`**. În momentul în care funcția se întoarce, valorile parametrilor poziționali sunt restaurate. Numele funcției este reținut în variabila `FUNCNAME`.

Variabila `sum` a fost definită în cadrul funcției și este vizibilă și în exterior. Dacă se dorește ca o variabilă să fie vizibilă numai în interiorul unei funcții aceasta trebuie prefixată folosind cuvântul cheie `local`.

12.10 Scripturile de pornire Bash

Scripturile de pornire Bash sunt fișiere intepretate la pornirea unei sesiuni shell. Acestea se ocupă de actualizarea variabilelor de mediu și de rularea unor comenzi personalizate de utilizator; un exemplu îl constituie configurarea de alias-uri.

Sesiunile de shell pot fi clasificate în:

- sesiuni **interactive/non-interactive** după cum permit sau nu rularea de comenzi prin prezentarea unui prompt utilizatorului; o sesiune non-interactivă este, de cele mai multe ori, folosită pentru rularea unui script shell
- sesiuni de **autentificare** (*login*)/**non-autentificare** (*non-login*); sesiunile de emulator de terminal (**`gnome-terminal`**, **`konsole`**, **`xterm`**) sunt sesiuni *non-login*

Fișierele de pornire sunt interpretate de sesiunile de shell interactive în modul următor:

- o sesiune de shell *login* va interpreta, în ordine, fișierele `/etc/profile`, apoi `~/.bash_profile`, `~/.bash_login` și `~/.profile`; la încheierea sesiunii se interpretează fișierul `~/.bash_logout`

- o sesiune de shell *non-login* va interpreta comenzile din `~/.bashrc`

  Fișierul `~/.bashrc` este fișierul recomandat pentru personalizarea sesiunilor de shell ale unui utilizator. Întrucât un shell de login nu interpretează `~/.bashrc`, fișierul `~/.bash_profile` va conține, de obicei, o linie de forma:

```
1 # include .bashrc if it exists
2 if [ -f ~/.bashrc ]; then
3     . ~/.bashrc
4 fi
```

12.10.1 Variabile de mediu

După cum a fost specificat anterior, unul din rolurile importante ale scripturilor de pornire shell este configurarea variabilelor de mediu.

Variabilele de mediu sunt variabile care afectează modul în care un proces rulează. În mod obișnuit, în sistemele Unix, un proces moștenește variabilele de mediu ale părintelui său. Astfel, pentru procesul asociat execuției unei comenzi, variabilele de mediu sunt cele ale shell-ului.

De cele mai multe ori, variabilele de mediu sunt configurate în scripturile de pornire ale Bash. Declarația unei variabile de mediu se face în formatul

```
1 export VARIABLE=value
```

Dacă nu se precizează **`export`**, atunci variabila nu va fi moștenită de procesele fiu. În mod obișnuit, variabilele de mediu sunt denumite cu majuscule.

Vizualizarea variabilelor de mediu existente la un moment dat se realizează cu ajutorul comenzii **`env`** sau **`set`**:

```
1 razvan@anaconda:~/uso/scripting$ env
2 SSH_AGENT_PID=7985
3 TERM=xterm
4 SHELL=/bin/bash
5  [...]
```

Exemple de variabile de mediu sunt:

- `PATH` reține o listă de directoare folosite de shell pentru a găsi executabilele asociate comenzii; lista este separată prin `:` (două puncte); dacă se dorește adăugarea directorului `~/bin` la variabila de mediu, se folosește comanda:

```
1 razvan@anaconda:~/uso/scripting$ echo $PATH
2 /usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
3
4 razvan@anaconda:~/uso/scripting$ export PATH="$PATH":~/bin
5
6 razvan@anaconda:~/uso/scripting$ echo $PATH
7 /usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games:/home/razvan/
  bin
```

- `HOME` reține directorul de bază al utilizatorului curent
- `USER` este numele utilizatorului curent
- `EDITOR` este editorul implicit

12.11 Studii de caz

12.11.1 Contorizarea numărului de utilizatori autentificați în sistem

Folosind cunoștințele din acest capitol, se poate rezolva corect problema contorizării numărului de utilizatori autentificați în ziua de *luni* în sistem, problemă propusă în secțiunea 12.3.4. Soluția este prezentată în continuare:

```
1  #!/bin/bash
2
3  user_day="Monday"
4
5  case $user_day in
6          "Monday") day_last="Mon";;
7          "Tuesday") day_last="Tue";;
8          "Wednesday") day_last="Wed";;
9          "Thursday") day_last="Thu";;
10         "Friday") day_last="Fri";;
11         "Saturday") day_last="Sat";;
12         "Sunday") day_last="Sun";;
13         *) echo "Zi invalida"; exit 1;;
14 esac
15
16 last | head -n -2 | while read user term ip day other; do
17         if test $day = $day_last; then
18                 echo "$user";
19         fi
20 done | sort -u | wc -l
```

Listing 12.7: Fișierul `count-login.bash`

```
1  razvan@anaconda:~/uso/scripting$ bash count-login.bash
2  9
```

Linia 3 definește variabila `user_day` folosită pentru a reține ziua despre care dorim informații (generalizat). Aceasta va fi dată în formatul complet pentru a avantaja utilizatorul. Liniile 5-14 se ocupă de traducerea zilei în formatul **`last`**; ziua în formatul **`last`** (prescurtat) este reținută în variabila `day_last`.

Liniile 16-20 reprezintă, de fapt, o comandă înlănțuită. Componentele lanțului de comenzii sunt:

- **`last`** – afișează informații despre utilizatorii autentificați în sistem;
- **`head -n -2`** – reține primele linii de la intrare, mai puțin ultimele două; este nevoie de acest lucru deoarece ultimele două linii date la ieșire de **`last`** nu sunt utile;
- ciclul `while` este folosit pentru a citi, folosind **`read`**, diversele câmpuri ale fiecărei linii date de last: numele de utilizator, terminalul asociat, adresa IP de conectare, ziua în care s-a realizat autentificarea; variabila `other`, fiind ultima variabilă, reține restul de informații până la sfârșitul liniei (informații care nu interesează); dacă ziua corespunde zilei reținute în variabila `day_last` atunci se afișează la ieșire numele de utilizator;

- **`sort -u`** – sortează și reține în mod unic liniile primite la intrare; în cazul nostru aceste linii sunt chiar cele afișate de ciclul `while`, adică utilizatorii care s-au autentificat în sistem în ziua dată de variabila `user_day`;
- **`wc -l`** – contorizează numărului de linii furnizat de **`sort -u`**; rezultatul este numărul de utilizatori distincți care s-au autentificat în sistem în ziua de dată de variabila `user_day`.

12.11.2 Schimbarea promptului shell

Promptul este șirul de caractere oferit de shell unde utilizatorul poate introduce comenzi. În Bash, un prompt care se termină în caracterul `$` reprezintă o sesiune de shell a unui utilizator neprivilegiat; un prompt care se termină în caracterul `#` este asociat unei sesiuni de shell a utilizatorului `root`.

Aspectul promptului este controlat prin intermediul unei variabile speciale, anume `PS1`. Aceasta controlează modul în care promptul este oferit utilizatorului. Comanda **`echo`** permite inspectarea variabilei:

```
1 razvan@anaconda:~$ echo $PS1
2 \u@\h:\w\$
```

Se observă că variabila cuprinde o serie de caractere speciale citate cu ajutorul *backslash*:

- `\u` înseamnă numele utilizatorului; în exemplul de mai sus se traduce în șirul `razvan`
- `\h` înseamnă numele sistemului
- `\w` înseamnă afișarea directorului curent, cu abrevierea directotului de bază la caracterul `~`
- `\$` înseamnă `#` pentru `root` sau `$` în rest

Promptul poate fi schimbat la un șir de caractere simplu:

```
1 razvan@anaconda:~$ PS1="prompt: "
2 prompt:
```

Promptul fi personalizat și folosind o serie de caractere speciale ca în exemplele de mai jos:

```
1  prompt: echo PS1="\u-\h[\d]\$ "
2  PS1=\u-\h[\d]$
3
4  prompt: PS1="\u-\h[\d]\$ "
5
6  razvan-anaconda[Fri Sep 14]$ PS1="\u+\h \W (\A)"
7
8  razvan+anaconda ~ (18:08)cd uso/
9
10 razvan+anaconda uso (18:08)cd ~/uso/conv/
11
12 razvan+anaconda conv (18:09)
```

Secvențe speciale care au fost folosite în exemplele de mai sus sunt:

- `\d` înseamnă data curentă
- `\A` înseamnă timpul curent
- `\W` înseamnă directorul curent ca și cale relativă față de directorul home (\w afișează calea completă)

O listă completă cu secvențele speciale de personalizare a promptului se poate găsi online[1].

Variabila `PS1` este definită în `/etc/profile` și `~/.bashrc`. Pentru configurarea permanentă a acesteia se recomandă utilizarea fișierului `~/.bashrc`.

Folosirea de culori în prompt

Promptul poate fi colorat prin folosirea secvenței speciale `\e[`. Astfel, dacă se dorește afișarea unui prompt de culoare roșie, se folosește comanda:

```
1 razvan@anaconda:~/uso/conv$ PS1="\e[0;31m\u@\h:\w\$ \e[m"
```

Pentru obținerea unui prompt de culoare verde comanda este:

```
1 razvan@anaconda:~/uso/conv$ PS1="\e[0;32m\u@\h:\w\$ \e[m"
```

Secvențele speciale `\e[x;ym` și `\e[m` definesc spațiul de aplicabilitate al unei culori. Secvența `x;y` definește culoarea utilizată. Secvențe posibile sunt:

- `0;30` – negru
- `0;31` – roșu
- `0;32` – verde
- `0;33` – galben
- `0;34` – albastru
- `0;35` – magenta
- `0;36` – cyan
- `0;37` – alb

În lista de mai sus `x` are valoare `0`. Se poate folosi valoarea `1` pentru versiunea deschisă a culorii. În mod evident, se pot asocia culori diferite pentru componentele promptului, ca în exemplul de mai jos:

```
1 razvan@anaconda:~$ PS1="\e[0;36m\u\e[m@\e[0;34m\h\e[m:\e[0;31m\w\e[m\$ "
```

În exemplul de mai sus, în cadrul promptului, numele utilizatorului va fi afișat folosind cyan, numele sistemului folosind albastru, iar numele directorului curent folosind roșu.

[1] http://tldp.org/HOWTO/Bash-Prompt-HOWTO/bash-prompt-escape-sequences.html

12.11.3 Batch scripting în Windows

În Windows, scripturile shell au ca echivalent *scripturi batch*. Un script batch are de obicei extensia `.bat`. Programul folosit pentru rularea scripturilor batch este **`cmd`** (interpretorul de comenzi pe Windows).

La fel ca un script shell, un script batch permite execuția de comenzi native Windows și programarea cu ajutorul variabilelor, a instrucțiunilor de decizie și a instrucțiunilor de ciclare.

Comenzi Windows

Rularea unei comenzi în Windows se realizează prin intermediul interpretorului de comenzi Windows. Pentru pornirea acestuia, în Windows XP trebuie pornită aplicația **Command Prompt**: *Start->All Programs->Accesories->Command Prompt*.

În afara comenzilor obișnuite pentru lucrul cu sistemul de fișiere (**`dir`**, **`cd`**, **`mkdir`**, **`rmdir`**, **`del`**, **`copy`**, **`move`** etc.), există și alte comenzi utile. Lista cu toate comenzile Windows se poate obține folosind comanda **`help`**:

```
C:\Documents and Settings\Razvan.RAGNAROK>help
For more information on a specific command, type HELP command-name
ASSOC    Displays or modifies file extension associations.

AT       Schedules commands and programs to run on a computer.

ATTRIB   Displays or changes file attributes.

BREAK    Sets or clears extended CTRL+C checking.

CACLS    Displays or modifies access control lists (ACLs) of files.

[...]
```

În tabelul 12.2 este prezentat un tabel cu comenzile utile Linux și echivalentul lor în Windows:

Operatori

Interpretorul de comenzi Windows are, asemănător interpretorului Bash, operatori de redirectare sau de comunicare între procese. În exemplul de mai jos dorim să reținem în fișierul `del.txt` informații despre comenzile care se ocupă cu ștergerea de informații:

```
C:\Documents and Settings\Razvan.RAGNAROK>help | findstr "[Dd]elete" >
del.txt

C:\Documents and Settings\Razvan.RAGNAROK>type del.txt
DEL      Deletes one or more files.

ERASE    Deletes one or more files.

LABEL    Creates, changes, or deletes the volume label of a disk.

```

```
10 [...]
```

Variabile

Ca și în Bash, variabilele nu au un tip. Definirea acestora se realizează cu ajutorul comenzii **`set`**:

```
1 C:\Documents and Settings\Razvan.RAGNAROK>set a=3
2
3 C:\Documents and Settings\Razvan.RAGNAROK>echo %a%
4 3
```

Afișarea conținutul unei variabile nu se mai realizează prefixând-o cu ajutorul caracterului `$`, ci folosind caracterul `%` înainte și după numele variabilei.

Ca și în Bash, comanda **`set`** este folosită și pentru afișarea variabilelor curente:

```
1  C:\Documents and Settings\Razvan.RAGNAROK>set
2  a=3
3  ALLUSERSPROFILE=C:\Documents and Settings\All Users
4  APPDATA=C:\Documents and Settings\Razvan.RAGNAROK\Application Data
5  CommonProgramFiles=C:\Program Files\Common Files
6  COMPUTERNAME=RAGNAROK
7  ComSpec=C:\WINDOWS\system32\cmd.exe
8  FP_NO_HOST_CHECK=NO
9  HOMEDRIVE=C:
10 HOMEPATH=\Documents and Settings\Razvan.RAGNAROK
11 [...]
```

Variabile de mediu Dacă în Bash variabilele de mediu sunt definite în fișierele de pornire, în Windows trebuie folosită o fereastră de configurare. Pentru accesul la aceasta se folosește: click dreapta pe *My Computer->Properties->Advanced->Environment Variables.*

Variabila **`PATH`** deține, ca și în cazul Bash, lista de directoare ale căror executabile pot fi rulate din linia de comandă. Spre deosebire de Bash, separatorul folosit este `;` (punct și virgulă).

În exemplul de mai jos se adaugă directorul `D:\mybin` la calea de executabile:

```
1 C:\Documents and Settings\Razvan.RAGNAROK>echo %PATH%
2 c:\program files\imagemagick-6.3.4-q16;C:\Program Files\MiKTeX 2.6\miktex
  \bin;C:
3 \WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;D:\Software\
  swftools
4
5 C:\Documents and Settings\Razvan.RAGNAROK>set PATH=%PATH%;D:\mybin\
6
7 C:\Documents and Settings\Razvan.RAGNAROK>echo %PATH%
8 c:\program files\imagemagick-6.3.4-q16;C:\Program Files\MiKTeX 2.6\miktex
  \bin;C:
9 \WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;D:\Software\
  swftools;D:\mybin\
```

Programarea batch

După cum s-a precizat, echivalentul scripturilor shell din Linux sunt scripturile batch din Windows. De obicei, acestea au extensia `.bat`. Exemplul cel mai simplu de script batch este prezentat mai jos:

```
C:\DOCUME~1\RAZVAN~1.RAG>type hw.bat
@echo off
echo "Hello, World!"

C:\DOCUME~1\RAZVAN~1.RAG>hw.bat
"Hello, World!"
```

În mod implicit, rularea unui script batch duce la afișarea comenzilor întâlnite (echivalent cu `set -x` din Bash). Pentru a preîntâmpina acest lucru se folosește directiva `@echo off`.

Un emulator al comenzii **`dir`** folosește `for` și este prezentat în continuare:

```
C:\DOCUME~1\RAZVAN~1.RAG>type dir.bat
@echo off
for %%f in (*) do echo %%f

C:\DOCUME~1\RAZVAN~1.RAG>dir.bat
.gtk-bookmarks
del.txt
dir.bat
gsview32.ini
hw.bat
logfile.txt
PUTTY.RND
winscp.RND
_viminfo
```

Instrucțiunea `if` are o sintaxa asemănătoare instrucțiunii echivalente din Bash:

```
C:\DOCUME~1\RAZVAN~1.RAG>set a=3

C:\DOCUME~1\RAZVAN~1.RAG>if %a% EQU 3 echo "true"
"true"
```

Mai multe despre comenzile instrucțiunile if și for se pot afla prin consultarea informației de ajutor: `if /?` și `for /?`.

12.11.4 PowerShell pe Windows

PowerShell este un nou interpretor de comenzi dezvoltat de Microsoft. Se bazează pe programare orientată obiect și pe versiunea 2.0 a platformei de dezvoltare .NET. Windows PowerShell este gratis; poate fi descărcat de pe site-ul asociat[1].

Windows PowerShell urmează, într-o oarecare măsură, ideea din spatele utilitarelor shell Unix, anume de a realiza sarcini complexe prin combinarea unor componente deja existente. Aceste componente sunt denumite cmdlets și sunt, de fapt, instanțe ale unor clase .NET. Deosebirea de shellurile Unix este faptul că în loc de a crea o bandă de transmitere a informației în format text (pipeline), cmdlet-ii comunică folosind obiecte.

[1] http://www.microsoft.com/windowsserver2003/technologies/management/powershell/default.mspx

Cmdlets

Cmdlet-ii reprezintă blocurile de bază pentru PowerShell echivalent cu comenzile de bază Unix folosite de Bash.

Astfel, pentru listarea proceselor din sistem se folosește cmdlet-ul **`get-process`**. Acesta prezintă aliasul **`ps`** și este echivalent comenzii **`ps`** din Linux. Pentru listarea conținutului unui director se folosește cmdlet-ul **`get-childitem`** cu aliasul **`ls`**.

Dacă, spre exemplu, un utilizator dorește omorârea proceselor al căror nume începe cu litera `p`, se folosește comanda:

```
PS> get-process p* | stop-process
```

Folosind concepte din programarea orientată obiect, se pot extrage câmpuri din rezultatul unei comenzi. Astfel, dacă se dorește aflarea timpului ultimului acces pentru un directorul `Cookies`, se folosește comanda:

```
PS> $(get-item Cookies).lastaccesstime
```

Pentru afișarea proceselor din sistem în formă tabelară se poate folosi comanda

```
PS> Get-Process | Format-Table Id, Name
  Id Name
  -- ----
2876 alg
 532 ApntEx
2044 Apoint
3448 calc
1824 CFSvcs
```

Variabile și structuri de control

PowerShell permite crearea de scripturi. Ca și Bash, printre facilitățile precizate se numără completare automată, utilizarea de variabile, structuri de control și funcții.

Spre deosebire de Bash, variabilele PowerShell au un tip:

```
PS C:\> $b = 3.1415926

PS C:\> $b
3.1415926

PS C:\> $b.GetType().Name
```

În Windows PowerShell, echivalentul pentru `for` (din Bash) este `foreach`:

```
PS C:\WINDOWS\system32> 1..10 | foreach { $_ * 2 }
2
4
6
8
10
12
14
16
18
20
```

Informații complete despre Windows PowerShell se găsesc în documentația care sosește cu pachetul de instalare.

Informații complete și tutoriale sed se pot găsi la online[1].

Cuvinte cheie

- shell
- bash
- script shell
- readline
- editarea comenzilor
- istoricul comenzilor
- reverse search
- autocompletion
- comenzi interne (built-in)
- comenzi externe
- shebang
- echo
- printf
- set
- exit
- read
- redirectarea intrării/ieșirii/erorii standard
- operatorii <, >, 2>, &>, >>
- here document, here string
- operatorii <<, <<<
- operatorii de secvențiere ;, &&, ||
- operatorul de comunicație |
- variabile shell
- caractere speciale shell: $, #, *, ", ‘ , \, (,), , , &, |, ;, [,], ?, blank
- if, test, comanda de test [...]
- for, while, until
- filtre de text: cat, tac, nl, sort, uniq, head, tail, cut, tr, grep
- expresie regulată (pattern matching)
- xargs, locate, find
- expandarea shell: expandarea variabilei, substituția comenzii, expandare aritmetică
- parametrii unui script shell; expresiile $0, $n, $#, $@, $?, $$, $!
- funcții shell
- scripturi de inițializare: .bashrc, .bash_profile, /etc/bash_profile
- variabile de mediu
- batch scripting
- power shell; cmdlet
- sed

Întrebări

1. Care din următoarele nu este un intepretor de comenzi?
   - ❑ Bash
   - ❑ PowerShell
   - ❑ Command Prompt

[1] http://sed.sourceforge.net/grabbag/tutorials/

- ❑ Flash

2. Ce comandă este folosită pentru căutarea unei expresii regulate într-un fișier?
   - ❑ tr
   - ❑ grep
   - ❑ cut
   - ❑ sort
3. Care din următorii operatori este asociat cu un *here document*?
   - ❑ `>`
   - ❑ `|`
   - ❑ `&>`
   - ❑ `<<`
4. Care din următoarele opțiuni testează egalitatea între două șiruri?
   - ❑ `set $a == $b`
   - ❑ `[$a == $b]`
   - ❑ `test $a == $b`
   - ❑ `{ $a == $b }`
5. Care este efectul comenzii **`sort -n file.txt`**?
   - ❑ sortează liniile nevide din fișier
   - ❑ sortează liniile vide din fișier
   - ❑ sortează liniile unice din fișier
   - ❑ sortează liniile în format numeric
6. Ce utilitar NU va ajuta un utilizator să substituie caracterul `a` cu `A` într-un fișier?
   - ❑ **`vim`**
   - ❑ **`cut`**
   - ❑ **`sed`**
   - ❑ **`tr`**
7. Care din următoarele NU este un avantaj al unui shell script?
   - ❑ viteza mare de execuție
   - ❑ automatizarea sarcinilor
   - ❑ folosirea de componente existente
   - ❑ depanare facilă
8. Care este utilitatea *shebang* (`#!`)?
   - ❑ permite editarea unui script shell

- ❑ permite asocierea de drepturi de execuție unui script shell
- ❑ prezintă calea către utilitarul folosit pentru rularea scriptului
- ❑ este o linie de comentariu specială într-un script shell

9. Care din următoarele NU este o instrucțiune în Bash?
   - ❑ `for`
   - ❑ `while`
   - ❑ `until`
   - ❑ `repeat`
10. Care din următoarele comenzi NU afișează intrările din directorul curent?
   - ❑ `ls`
   - ❑ `for i in *; do echo $i; done`
   - ❑ `find .  -mindepth 2 -type f`
   - ❑ `ls *`

Tabelul 12.1: Condiții în shell

operator	tip operanzi	condiție verificată	utilizare
`-e`	1 fișier	fișierul există	`-e fisier`
`-f`	1 fișier	fișierul este fișier obișnuit (regular file)	`-f fisier`
`-s`	1 fișier	fișierul are dimensiune non-zero	`-s fisier`
`-d`	1 fișier	fișierul este director	`-d fisier`
`-r`	1 fișier	fișierul poate fi citit	`-r fisier`
`-w`	1 fișier	fișierul poate fi scris	`-w fisier`
`-x`	1 fișier	fișierul poate fi executat	`-x fisier`
`-O`	1 fișier	fișierul este deținul de utilizator	`-O fisier`
`-G`	1 fișier	grupul ce deține fișierul este și grupul utilizatorului	`-G fisier`
`-nt`	2 fișiere	primul fișier este mai recent decât al doilea	`fisier1 -nt fisier2`
`-ot`	2 fișiere	primul fișier este mai puțin recent decât al doilea	`fisier1 -ot fisier2`
`-eq`	2 numere	numerele sunt egale	`numar1 -eq numar2`
`-ne`	2 numere	numerele nu sunt egalee	`numar1 -nq numar2`
`-gt`	2 numere	primul număr este mai mare decât al doilea	`numar1 -gt numar2`
`-ge`	2 numere	primul număr este mai mic sau egal cu al doilea	`numar1 -ge numar2`
`-lt`	2 numere	primul număr este mai mic decât al doilea	`numar1 -lt numar2`
`-le`	2 numere	primul număr este mai mic sau egal cu al doilea	`numar1 -le numar2`
`=`	2 șiruri de caractere	șirurile sunt identice	`sir1 = sir2`
`!=`	2 șiruri de caractere	șirurile nu sunt identice	`sir1 != sir2`
`-z`	1 șir de caractere	șirul este vid	`-z sir`

Tabelul 12.2: Comenzi utile Linux și Windows

Linux	Windows	Efect
pwd	cd	afișează directorul curent
ls	dir	listează conținut director
cd	cd	schimbă director
mkdir	mkdir, md	creează director
rmdir	rmdir, rm	șterge director
touch	–	creează fișier
rm	del, erase	șterge fișier
cp	copy	copiază fișier
mv	move	mută fișier
cat	type	afișează conținut fișier
echo	echo	afișează mesaj
less, more	more	afișare paginată fișier
cmp	comp	compară fișiere
diff	fc	compară și afișează diferențe
file	ftype	afișează tipul fișierului
grep	findstr	caută în fișier
shift	shift	schimbă parametrii poziționali
sort	sort	sortează intrarea

Capitolul 13

Mediul grafic

A bus station is where a bus stops. A train station is where a train stops. On my desk I have a workstation...
(Anonymous)

Ce se învață din acest capitol?

- Tipuri de imagini, fonturi, Unicode, UTF-8
- Interfața grafică în Linux: X Window System
- Componentele X Window System: window manager, desktop environment, display manager
- Configurări de bază: rezoluția, aranjamentul caracterelor pe tastatură
- Configurarea sistemului folosind KDE
- Servicii desktop
- Configurare X peste SSH, VNC, RDC

În momentul de față, cea mai utilizată metodă de interacțiune cu sistemul de operare este interfața GUI. Principalul său avantaj față de celelalte tipuri de interfețe (CLI, TUI) (vezi și secțiunea 2.3.1) constă în ușurința cu care utilizatorii învață să o folosească.

Evoluția interfețelor grafice s-a realizat în paralel cu evoluția hardware-ului. Prima încercare de realizare a unei interfețe grafice a avut loc în anii '60 la *Institutul de Cercetări al Universității Standford*[1]. La acel moment, interfața grafică cuprindea un mouse elementar cu ajutorul căruia se puteau accesa link-uri în documentele text (*text hyperlinks*).

În momentul de față, interfețele grafice încep să păsească în era 3D. Aero din Windows Vista/7[2], Compiz[3]/KWin[4] din Linux și Quartz Compositor[5] din MacOS X oferă

[1] Stanford Research Institute – http://www.sri.com/

[2] http://www.microsoft.com/windows/windows-vista/features/aero.aspx
http://www.microsoft.com/windows/windows-7/features/aero.aspx

[3] http://en.wikipedia.org/wiki/Compiz

[4] http://en.wikipedia.org/wiki/KWin

[5] http://arstechnica.com/apple/reviews/2005/04/macosx-10-4.ars/13

utilizatorului o experiență 3D la a cărei realizare participă în mare măsură placa video și nu procesorul ca până acum.

13.1 Concepte în mediul grafic

13.1.1 Tipuri de imagini

În funcție de modul de stocare a informației, imaginile digitale se impart în două tipuri: *imagini raster* (bitmap) și *imagini vectoriale*.

> O imagine **raster** (*bitmap*) reține informația ca o matrice de puncte.

De exemplu, fotografiile realizate cu o cameră foto digitală sunt imagini raster. Este formatul de imagini cel mai des folosit, având diverse implementări: `BMP`, `JPEG`, `GIF`, `PNG` etc. Dezavantajul acestui tip de imagini este că, atunci când imaginea este redimensionată (scalată), conținutul noii imagini trebuie aproximat din conținutul imaginii inițiale. Această aproximare se poate realiza cu rezultate bune în cazul în care imaginea este micșorată. În cazul în care imagine este mărită, rezultatele sunt cu atât mai nesatisfăcătoare cu cât factorul de scalare este mai mare, precum se poate vedea în figura 13.1.

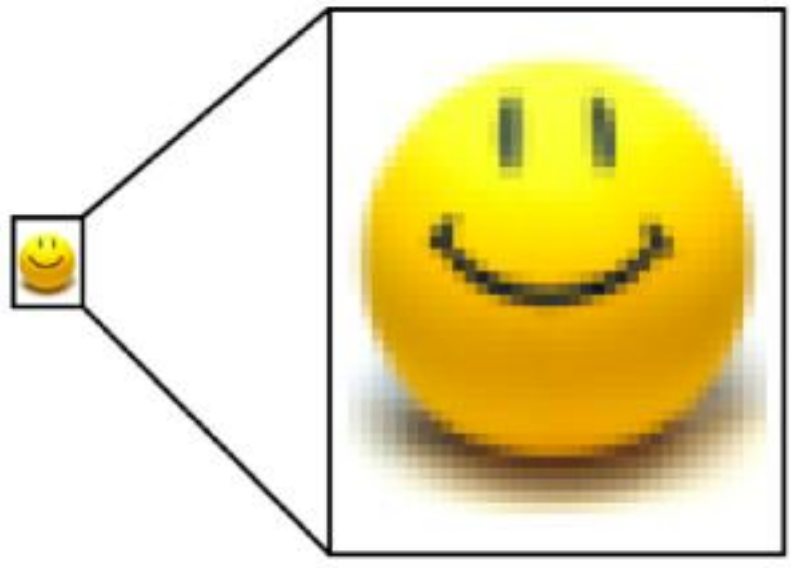


Figura 13.1: Mărirea unei imagini în format raster

Fiecare punct dintr-o imagine raster (bitmap) este descris prin mai multe componente, în funcție de suportul de ieșire:

- **pe ecran** un punct dintr-o imagine raster este descris prin 3 valori reprezentând intensitatea luminoasă (pe 8 biți) a următoarelor culori: *roșu* (Red), *verde* (Green), *albastru* (Blue) – prescurtate RGB. În cazuri speciale (prelucrări profesionale de imagini), intensitatea fiecărei culori poate fi exprimată pe 16 biți (crescând astfel spectrul de culori ce poate fi reprezentat). De asemenea, pe langă cele 3 componente (RGB), anumite imagini digitale (precum formatul PNG) conțin și o a 4-a componentă numită *transparență* (Alpha). Astfel, în cazul imaginilor cu transparență (RGBA), pentru fiecare punct se asociază și un nivel de transparență, permițând astfel ascunderea (totală sau parțială) a unor părți din imagine.

- **pe suport tipărit**, un punct dintr-o imagine raster este descris prin 4 valori, reprezentând cantitatea de culoare necesară pentru obținerea culorii punctului din patru componente: cyan (C), magenta (M), galben (Yellow), negru (black – K)

Într-o imagine **vectorială** informațiile sunt reținute sub formă elemente geometrice de bază: linii, elipse, puncte și gradienți de culoare.

Avantajul major al acestor tipuri de imagini este acela că sunt foarte ușor de redimensionat (practic, culoarea fiecărui punct din imagine poate fi calculată exact pe baza elementelor ce compun imaginea). În urma redimensionării (micșorare sau mărire), calitatea imaginii nu se pierde (figura 13.2). Există câteva formate mai cunoscute care stochează conținutul în format vectorial, cele mai cunoscute fiind: `SVG`[1], `PS`, `EPS`, `PDF`.

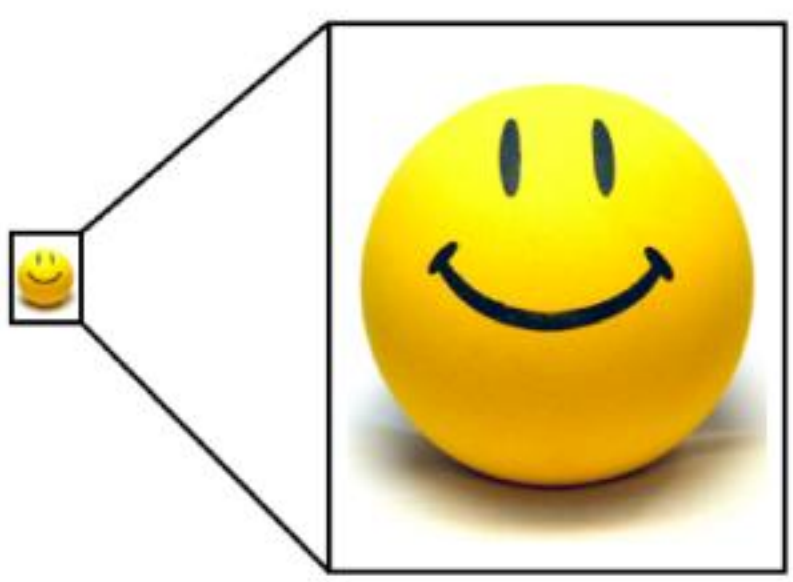


Figura 13.2: Mărirea unei imagini în format vectorial

13.1.2 Fonturi. Unicode

Fonturile sunt fișiere într-un anumit format care conțin informații despre modul de desenare a caracterelor pe ecran. Printre cele mai utilizate în prezent formate pentru fonturi se numără *True Type* (`.ttf`) și *Open Type* (`.otf`).

Fonturile sunt de cele mai multe ori descrieri vectoriale ale aspectului caracterelor, motiv pentru care ele pot scala atât la dimensiuni foarte mici cât și la dimensiuni foarte mari fără pierderi calitative. Exista 4 stiluri mai cunoscute în care se poate prezenta un font: *normal*, *italic* (caractere aldine – caracterele sunt ușor înclinate spre dreapta), *bold* (caracterele sunt îngroșate), *bold-italic* (caractere aldine îngroșate).

Inițial, setul de caractere ASCII[2] a necesitat 7 biți pentru reprezentare (un bit fiind folosit pentru calculul parității în transmiterea fiecărui caracter). Acest lucru permitea existența a 128 caractere, 33 dintre acestea fiind caractere de control. Ulterior, când bitul 8 nu a mai fost folosit ca bit de paritate, folosirea acestuia a dus la extinderea setului de caractere cu încă 128. Pe cele 128 de noi poziții au fost plasate caractere și simboluri dar s-a constatat necesitatea unor extensii.

[1] acceptat de W3C ca standard pentru WWW – http://www.w3.org/Graphics/SVG/

[2] http://en.wikipedia.org/wiki/ASCII

Astfel, a apărut noțiunea de pagină de coduri (code page)[1] prin care erau definite noile 128 de caractere disponibile. Pentru o perioadă de timp s-au folosit frecvent pagini de coduri create fie pentru o anumită limbă fie pentru grupuri de limbi.

Nevoia de a avea mai mult de 256 caractere accesibile la un moment dat a dus la impunerea unui nou standard de reprezentare a setului de caractere – **Unicode** (în prezent ajuns la versiunea 5). Acesta permite reprezentarea prin până la 4 octeți a unui caracter (folosind maxim 32 de biți). În acest fel sunt acoperite inclusiv caracterele asiatice.

Există mai multe standarde[2] pentru codificarea caracterelor Unicode în fișiere. Cel mai folosit este `UTF-8`, format în care primele 128 de caractere sunt identice cu cele din ASCII. Acesta este un format cu număr variabil de octeți, numărul de octeți ocupați de un caracter fiind dat de numărul de biți de `1`, până la primul bit `0`, din primul octet (începând de la cel mai semnificativ bit[3]). Dacă primul bit este `0`, atunci caracterul ocupă un octet și, în mod special, este acelasi caracter ca în setul ASCII. Dacă un octet are primii biți `10`, atunci este un octet care continuă reprezentarea unui caracter. Mai multe informații despre codificarea `UTF-8` se pot găsi online[4].

În prezent, fonturile importante din sistemele de operare moderne au suport pentru un număr semnificativ de caractere definite în standardul Unicode.

13.2 Interfața grafică în Linux. Componente

13.2.1 X Window System

> **X Window System** (cunoscut ca **X11** sau ca **X**) este un cadru (framework) ce permite dezvoltarea de interfețe grafice în medii Unix.

Monitoarele (CRT sau LCD) afișează imagini în format raster. Din acest motiv, imaginea finală trimisă către acestea este în format raster.

X Window System permite: desenarea și mutarea ferestrelor pe ecran, interacțiunea cu mouse-ul și interacțiunea cu tastatura. Restul este lăsat în seama programelor, motiv pentru care aspectul vizual al interfețelor grafice bazate pe X diferă destul de mult de la un sistem de operare la altul, sau chiar între versiunile aceluiași sistem de operare.

X a fost creat în 1984 la MIT[5]. Versiunea curenta a protocolului este X11 și a apărut în 1987. În momentul de față proiectul X este condus de către X.Org Foundation și a ajuns la versiunea X11 Release 7.4.

---

[1] http://en.wikipedia.org/wiki/Code_page
[2] http://en.wikipedia.org/wiki/Comparison_of_Unicode_encodings
[3] http://en.wikipedia.org/wiki/Most_significant_bit
[4] http://en.wikipedia.org/wiki/UTF-8
[5] http://www.mit.edu

13.2.2 Arhitectura X Window System

X Window System are o arhitectură de tip client-server: mai mulți clienți X se conectează la un server X (figura 13.3). Serverul acceptă cereri de desenare a obiectelor grafice și trimite înapoi către clienți intrările (inputul) de la utilizator (preluate de la tastatură, mouse etc.).

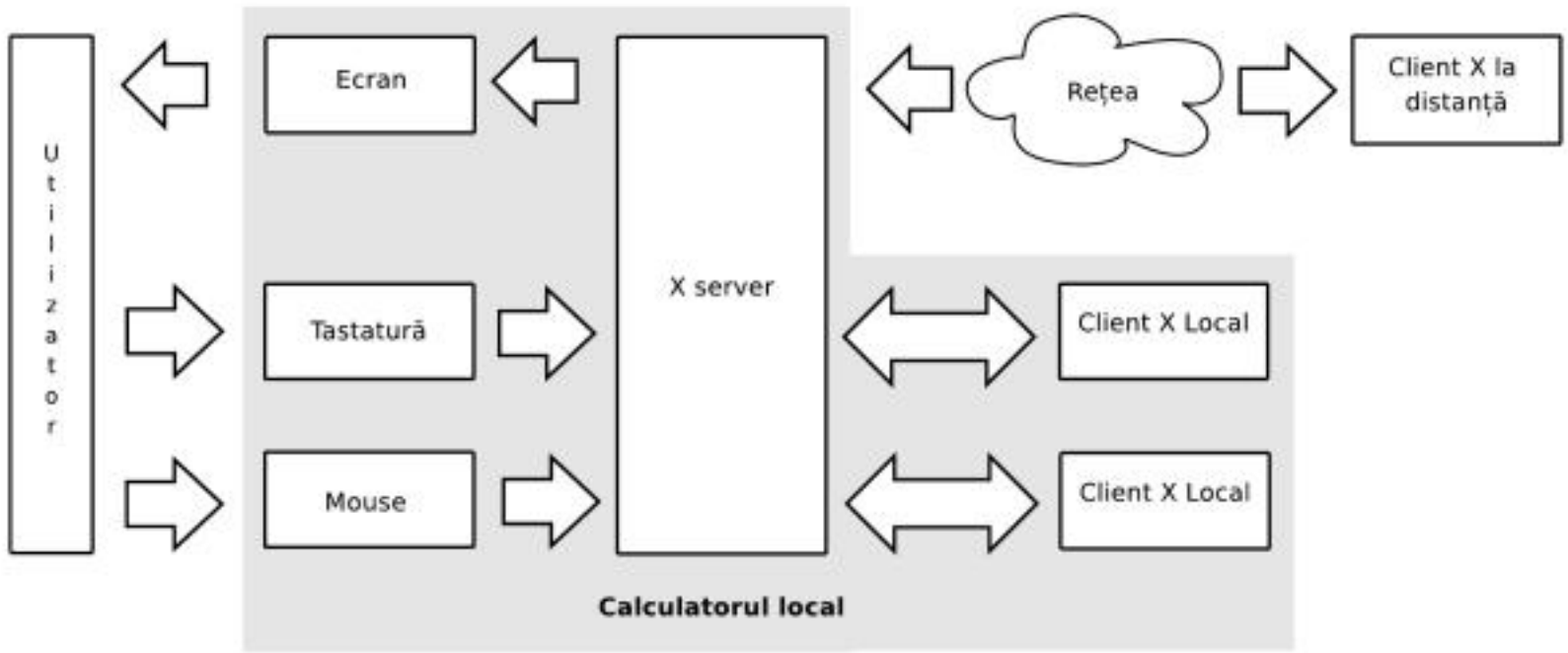


Figura 13.3: Arhitectura X Window System

Serverul X este responsabil de administrarea resurselor ce permit interacțiunea cu utilizatorul. Aceste resurse sunt: *ecranul*, *tastatura*, *mouse-ul*, *touchpad-ul*, *stylus-ul*, *joystick-ul* etc. Serverul X rulează pe calculatorul care interacționează cu utilizatorul și trimite intrările (inputul) primit de la acesta către clienți. Clienții procesează informația primită de la server și îi trimit acestuia înapoi informații pe care serverul le va afișa utilizatorului pe ecran. În prezent, cea mai utilizată implementare a server-ului X este oferită de proiectul X.Org[1].

Clienții X se conectează la un server X pentru a folosi resursele administrate de acesta. Ei cer serverului să afișeze pe ecran obiecte grafice și preiau de la acesta intrările (inputul) utilizatorului, precum apăsările de taste sau mișcările mouse-ului. Clienții pot rula local (pe același calculator cu serverul) sau pot rula pe un alt calculator aflat la distanță. Exemple de clienți X: un browser web, un client de e-mail, un program de mesagerie instant.

Protocolul X este protocolul utilizat în comunicația dintre clienți X și serverul X.

Comunicația dintre clienți și server se desfășoară diferit față de modelul general al comunicației client-server. În general clientul, ca localitate, se află pe calculatorul local iar serverul se află pe un calculator aflat la distanță. În cazul *X Window System* lucrurile stau invers. Serverul se află pe calculatorul local pentru a putea trimite și primi informații de la utilizator. Clienți se pot afla tot pe calculatorul local sau pe unul aflat la distanță.

În exemplul din figura 13.4, pe un calculator aflat la distanță rulează trei clienți X. Acest calculator poate să nu aibă conectat niciun monitor și nici mouse sau tastatură. Fiecare din cei trei clienți este conectat la un alt server X (fiecare server X fiind pe un calculator

[1] http://www.x.org/

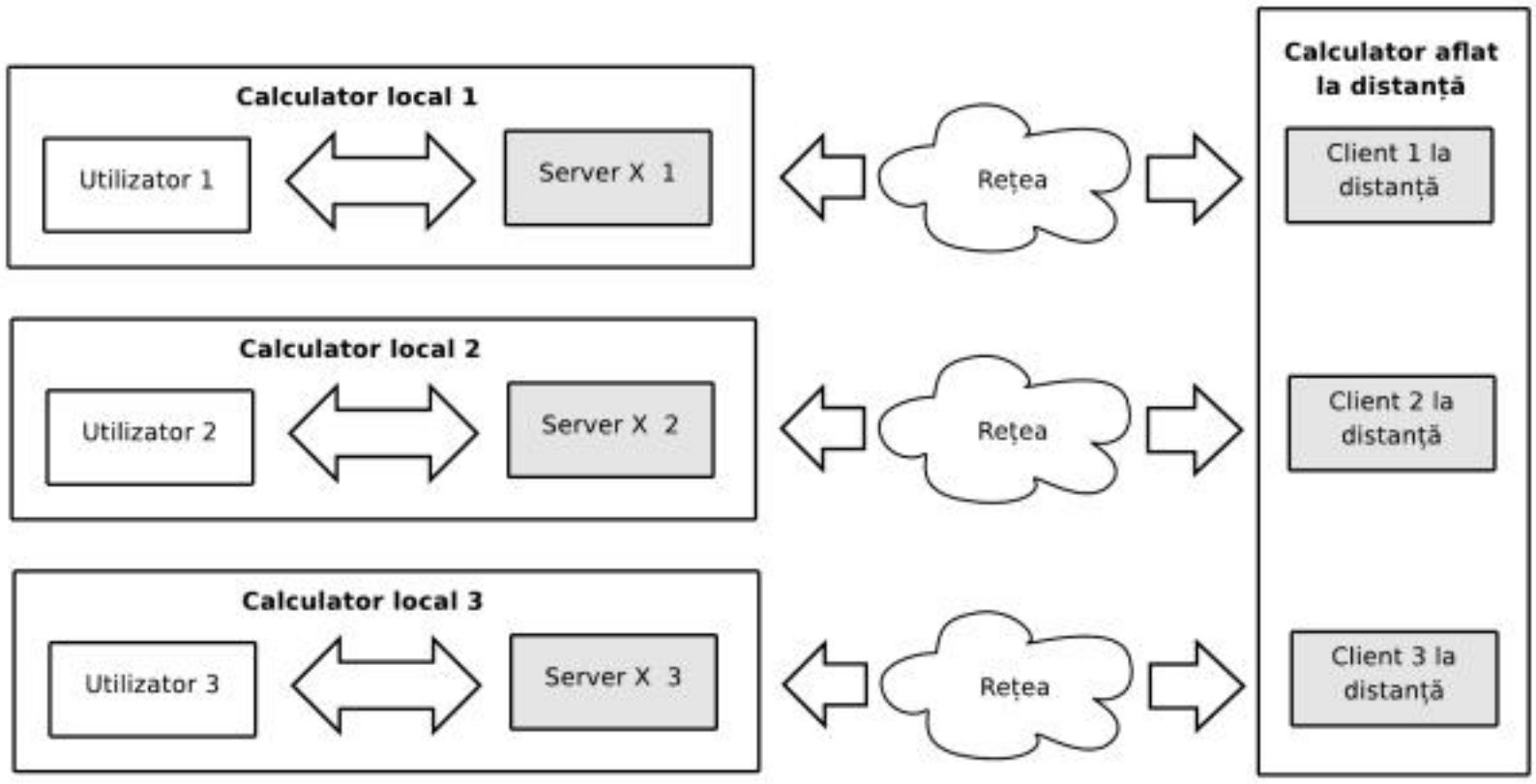


Figura 13.4: Exemplu de conectare client-server folosind protocolul X

diferit). Fiecare server X preia datele de la utilizatorul local (de exemplu prin intermediul tastaturii) și le transferă clientului, primind înapoi de la acesta informații care trebuie prezentate utilizatorului pe ecran. Toată procesarea informației de la utilizator este realizată de calculatorul aflat la distanță (acolo unde se execută clientul). Practic, în modelul de mai sus, calculatorul aflat la distanță se ocupa de procesarea datelor, calculatoarele 1, 2 și 3 funcționând pe post de terminale.

X nu conține nicio specificație legată de felul în care arată interfața cu utilizatorul (*Application User Interface*): modul în care arată butoanele, meniurile, ferestrele etc. Aceste specificații sunt implementate în schimb de către alte componente, precum *Window Manager*-ul (manager-ul de ferestre), *Desktop Environment*-ul (mediul desktop) sau de bibliotecile de componente grafice utilizate de aplicație aplicație.

Un window manager controlează amplasarea și modul în care arată ferestrele aplicațiilor. Acesta este un tip special de client X (se conectează la serverul X ca orice alt client) care controlează felul în care arată ceilalți clienți X.

> **Window manager**-ul face ca toți clienții să aibă caracteristici comune: bara de titlu a ferestrei, butoanele de minimize și maximize etc. El poate crea o interfață asemănătoare cu cea din Microsoft Windows sau cu cea din Mac OS X.

Window managerele pot oferi funcționalități de bază (precum `twm`[1], window managerul care vine împreună cu X) sau funcționalități complexe (cum ar fi Enlightenment ce se apropie ca funcționalitate de un desktop environment). Printre cele mai cunoscute window managere se numără: *Kwin*[2] este window managerul folosit de KDE, *Metacity*[3] cel folosit de GNOME și *Compiz*[4] este cel utilizat pentru desenarea interfeței 3D din proiectul cu același nume.

[1] http://en.wikipedia.org/wiki/Twm
[2] http://en.wikipedia.org/wiki/Kwin
[3] http://en.wikipedia.org/wiki/Metacity
[4] http://en.wikipedia.org/wiki/Compiz

În cazul în care această componentă lipsește, se vor putea crea ferestre noi dar nu vor avea bara de titlu. În plus, ele nu vor putea fi mutate pe ecran, nu vor putea fi închise cu `ALT+F4`, nu va funcționa `ALT+TAB`, nu se va putea muta o fereastră din fundal (*background*) în prim-plan (*foreground*). În schimb, conținutul ferestrelor va rămâne neschimbat.

Stilul în care arată ferestrele (aspectul lor vizual) este controlat de o componentă a window managerului numită **window decorator**. Acesta este responsabil cu desenarea titlului ferestrei, a butoanelor și a marginii acesteia.

Un **desktop environment** (mediu desktop) include un window manager, mai multe aplicații și o interfață consistentă din punct de vedere vizual.

Cele mai cunoscute medii desktop sunt KDE[1] (figura 13.5) și GNOME[2] (figura 13.6).

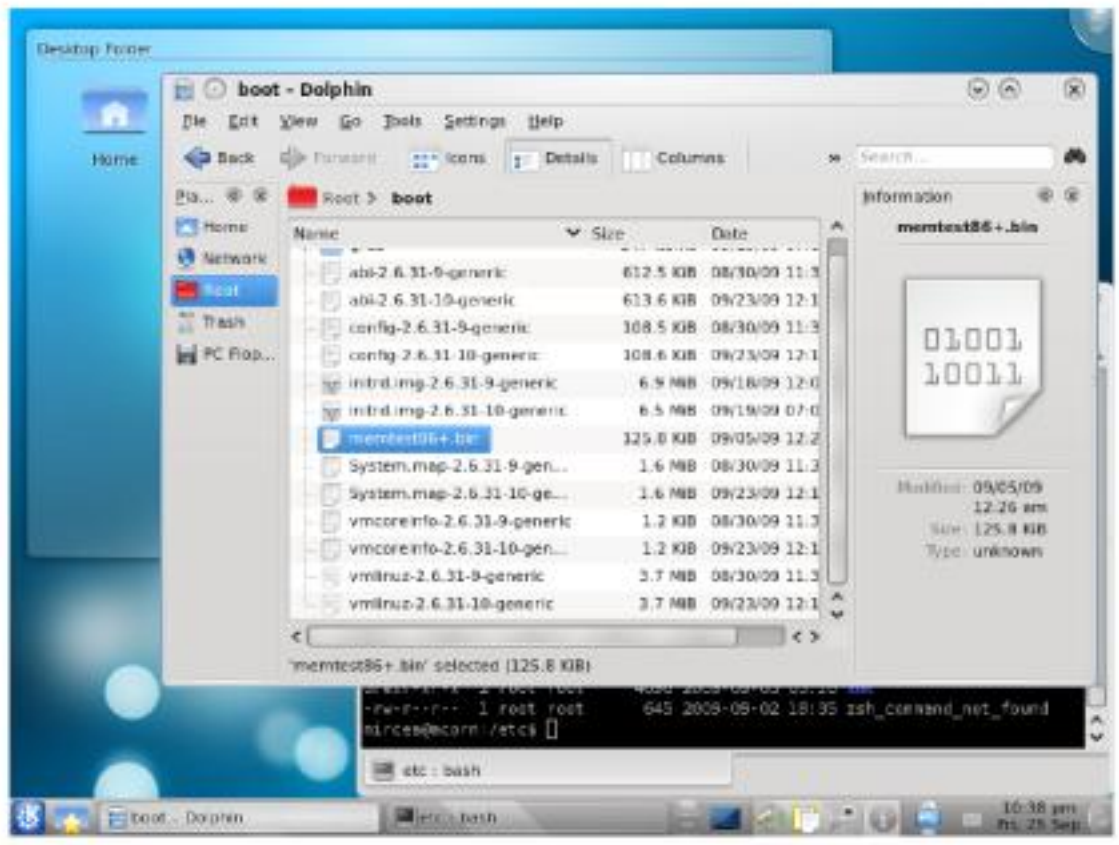


Figura 13.5: KDE

Un **display manager** este un program care permite pornirea unei sesiuni pe un server X de pe același calculator sau de pe unul aflat la distanță. Acesta este implementat ca un tip special de client X.

Display managerul afișează utilizatorului un ecran de autentificare (*login*) permițându-i acestuia să introducă un nume de utilizator și o parolă. După ce autentificarea s-a realizat cu succes o nouă sesiune este pornită. Exemplu de display managere: `kdm` (figura 13.7) folosit de KDE, `gdm` (figura 13.8) folosit de GNOME sau `xdm`, display managerul implicit din X Window System.

Atunci când display managerul rulează pe același calculator cu serverul X, el va porni serverul înainte de afișa fereastra de autentificare. Astfel display manager-ul are o funcție asemănătoare cu procesele `login` și `getty`(vezi secțiunea 6.5.3).

[1] http://en.wikipedia.org/wiki/KDE
[2] http://en.wikipedia.org/wiki/GNOME

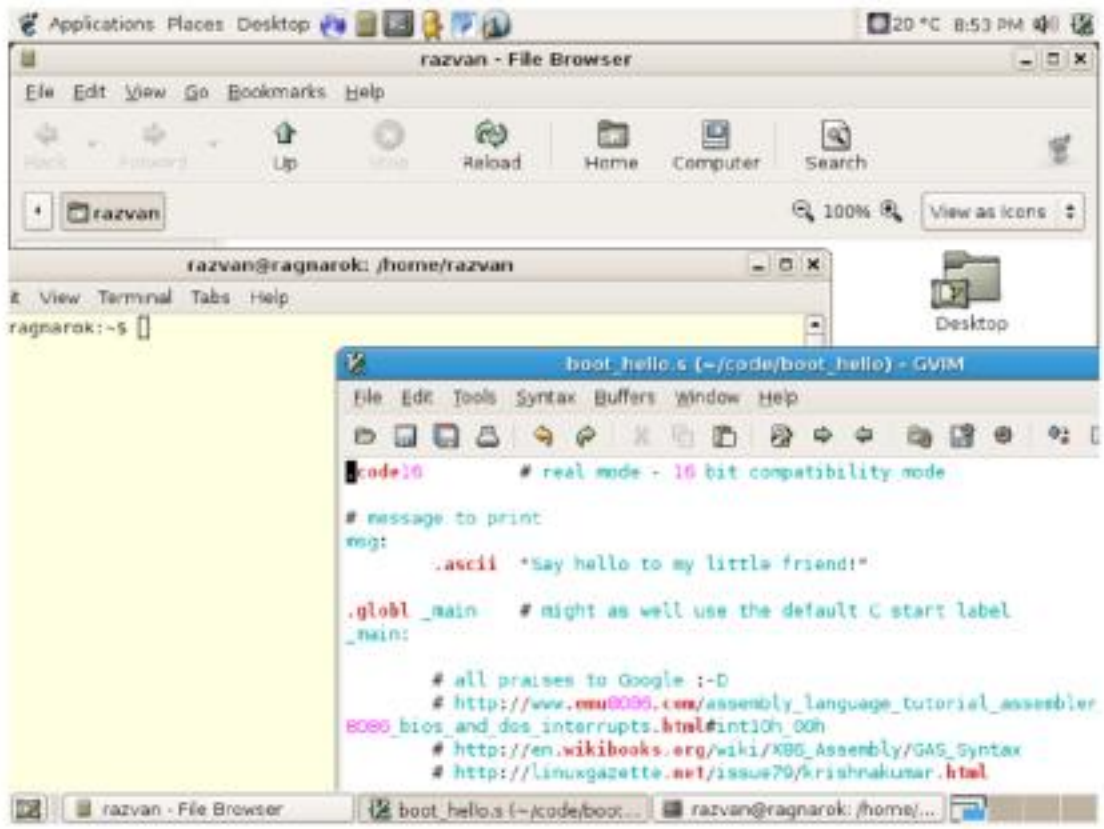


Figura 13.6: GNOME

Figura 13.7: Fereastra de login din Kubuntu (KDM)

13.3 Pornirea și oprirea interfeței grafice

În cazul în care interfața grafică nu pornește odată cu sistemul de operare, ea poate fi pornită separat. Pentru a realiza acest lucru există mai multe posibilități:

Folosind `xinit`

`xinit` este un program care pornește serverul X și un prim client care se conectează la el. Utilizatorul nu mai trebuie să se autentifice într-o fereastră de login deoarece el s-a autentificat în sistem înainte de a rula comandă **`xinit`**. Atunci când acest prim client își încheie activitatea, **`xinit`** va opri serverul X:

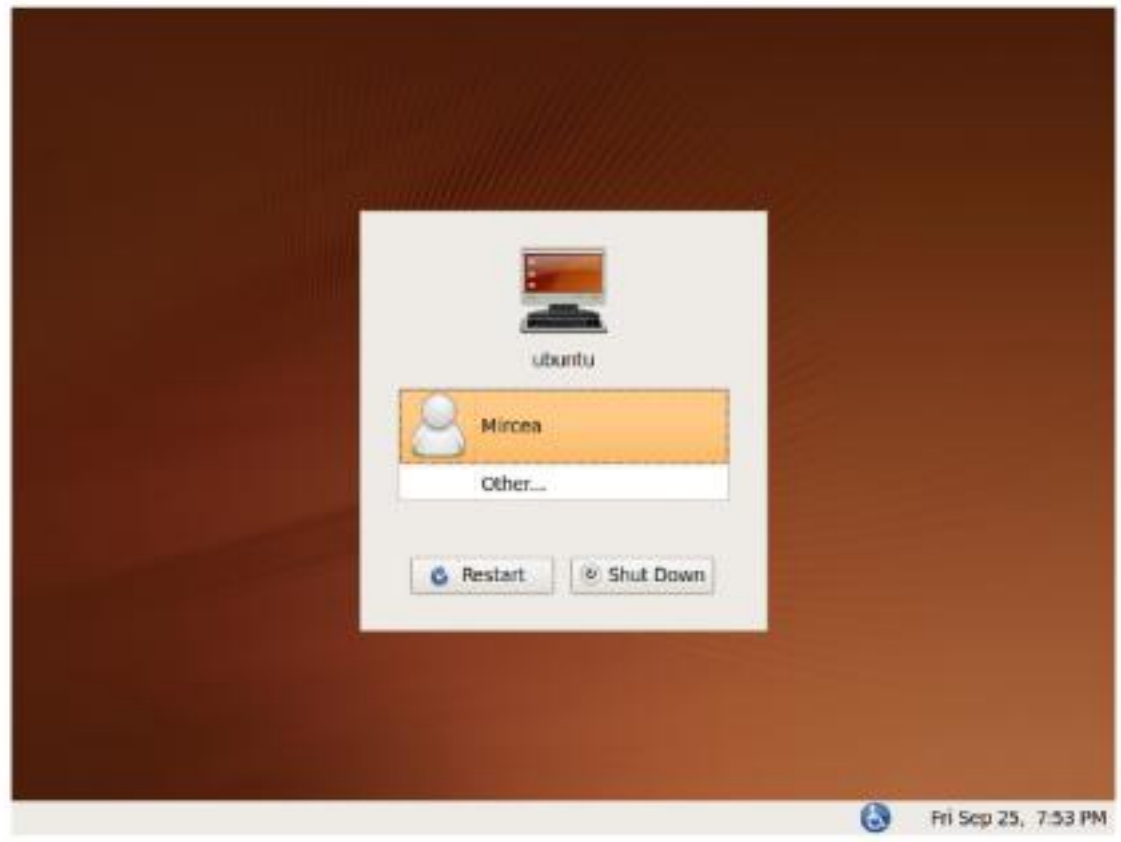


Figura 13.8: Fereastra de login din Ubuntu (GDM)

```
1 george@asgard:~$ man xinit
2 [...]
3 The xinit program is used to start the X Window System server and a first
   client program on systems that cannot start X directly from /etc/init or
   in environments that use multiple window systems.  When this first
  client exits, xinit will kill the X server and then terminate.
```

Folosind `startx`

`startx` este un front-end pentru **`xinit`** care oferă utilizatorului o interfață mai complexă:

```
1 george@asgard:~$ man startx
2 [...]
3 The startx script is a front end to xinit that provides a somewhat nicer
  user interface for running a single session of the X Window System. It is
   often run with no arguments.
```

Folosind un Display Manager

Atunci când un Display Manager este pornit pe calculatorul local, el va porni mai întâi un server X, urmând ca apoi să prezinte utilizatorului un ecran de autentificare. Dupa autentificare este pornit un server X în numele utilizatorului.

Pornirea unui Display Manager se poate face în mod asemănător serviciilor:

```
1 root@asgard:/home/george# service kdm start
```

Pentru **`gdm`** se procedează în mod asemănător.

Pentru oprirea interfeței grafice este suficientă oprirea display managerului. Acest lucru se poate realiza în felul următor:

```
1 root@asgard:/home/george# kill -9 *dm
```

Comanda anterioară oprește toate procesele al căror nume este oricât de lung și se termină cu șirul de caractere "`dm`".

O alta metodă de oprire a interfeței grafice este oprirea serviciului `kdm` (sau `gdm`) folosind paramerul `stop`:

```
root@asgard:/home/george# service kdm stop
```

Pentru oprirea și repornirea imediată a interfeței grafice este suficient să fie repornit display managerul folosind serviciile `kdm` (sau `gdm`) însă cu parametrul `restart`:

```
root@asgard:/home/george# service kdm restart
```

De asemenea oprirea și repornirea imediată a serverului X se poate realiza și prin combinația de taste `Ctrl+Alt+Backspace`.

13.4 Configurarea serverului X

Această secțiune se va concentra pe prezentarea modurilor de configurație temporară a server-ului X. KDE și GNOME vin cu aplicații pentru configurarea permanentă a server-ului X, salvând configurațiile în fișiere specifice Desktop Environment-ului.

În ultimele versiuni, necesitatea utilizării unui fișier de configurare a server-ului X a fost eliminată. Fișierul de configurare utilizat în versiunile anterioare se numea `/etc/X11/xorg.conf`.
Principalul dezavantaj al acestuia era faptul că trebuia repornit server-ul la fiecare modificare a fișierului. Fișierul mai poate fi utilizat în prezent pentru realizarea configurărilor permanente, dar, pentru simplitate în configurare, comenzile pentru configurarea temporară se pot salva în fișierul `~/.xprofile`.

Fișierul `/etc/X11/xorg.conf`, dacă este folosit, trebuie structurat pe secțiuni specifice fiecărui dispozitiv, după cum se poate observa și online[1] sau în documentația **man** (**`man xorg.conf`**).

13.4.1 Configurarea rezoluției

Versiunile recente ale server-ului X implementeaza extensia `Xrandr`, extensie care permite modificarea live (fără repornirea mediului grafic) a rezoluției, dimensiunii desktop-ului, orientării ecranelor etc. Acest lucru reprezintă un mare avantaj în cazul sistemelor la care se conecteaza monitoare externe (precum laptop-urile).

Extensia `Xrandr` poate fi utilizata folosind comanda **`xrandr`**. Câteva utilizări frecvente ale comenzii sunt prezentate mai jos:

- afișarea stării curente a sistemului grafic

```
mircea@acorn:~$ xrandr
Screen 0: minimum 320 x 200, current 1280 x 1024, maximum 1280 x
1280
VGA connected 1280x1024+0+0 (normal left inverted right x axis y
axis) 338mm x 270mm
   1280x1024      60.0*+   75.0     60.0*
   1152x864       75.0
```

[1] https://wiki.ubuntu.com/X/Config/Resolution

```
   1024x768        75.0     70.1     60.0
   832x624         74.6
   800x600         72.2     75.0     60.3     56.2
   640x480         75.0     72.8     66.7     59.9
   720x400         70.1
```

- activarea tuturor monitoarelor conectate la rezoluția lor default:

```
mircea@acorn:~$ xrandr --auto
```

- dezactivarea unui monitor (LCD-ul în acest caz):

```
mircea@acorn:~$ xrandr --output LVDS --off
```

- activarea unei anumite rezoluții (se poate preciza opțional și rata de refresh – pentru 75 Hz se folosește parametrul `--rate 75`)

```
mircea@acorn:~$ xrandr --output LVDS --mode 1024x768
```

13.4.2 Configurarea tastaturii

Printre cele mai importante configurări ale tastaturii se numără aranjamentul caracterelor pe taste (*keyboard layout*). Această configurație este importantă mai ales în condițiile în care un utilizator dorește să scrie caractere care nu se găsesc implicit pe tastatură.

În limba română s-au impus mai multe standarde de-a lungul timpului. Unele presupuneau înlocuirea completă a unor caractere de pe tastatură pentru includerea diacriticelor. În 2004 a fost decis un nou standard pentru aranjamentul caracterelor românești pe tastatură, cu următoarele specificații (pe scurt):

- caractere românești sunt disponibile ca aranjament secundar pentru tastatura US, apăsând tasta `AltGr` (tasta `Alt` din dreapta) – majoritatea tastaturilor din România au aranjamentul US, foarte puține fiind inscripționate cu diacritice românești în vechiul standard
- diacriticele `ă`, `î`, `ș`, `ț` sunt disponibile apăsând `AltGr` și aceleași taste dar fără virgulă (respectiv `a`, `i`, `s`, `t`)
- `â` este accesibil pe tasta `q`
- pentru a obține literele mari se apasă și tasta `Shift`

Astfel, pentru a obține caracterul `Ț` se apasă tastele: `AltGr+Shift+t`; pentru a obține caracterul `â` se apasă `AltGr+q`.

Pe lângă acestea, aranjamentul permite și obținerea caracterului €(euro) prin apăsarea combinației de taste `AltGr+e`.

O altă problemă în cazul aranjamentului de tastatură pentru limba română este modul în care se formează diacriticele `ș` și `ț`. Mulți utilizatori nu remarcă faptul că aceste două caractere (și literele mari echivalente) au două reprezentări posibile: *cu virgulă* (virgula este la o mică distanță de caracter) și *cu sedilă* (virgula este lipită de caracter). Standardul realizat de *Academia Română* menționează că diacriticele se construiesc folosind virgulă. O mare parte din implementările de aranjamente din sistemele de operare sunt greșite din acest punct de vedere.

Pentru a activa aranjamentul de tastatură pentru limba română conform standardului, în Linux se utilizează comanda:

```
1 mircea@acorn:~$ setxkbmap ro
```

Varianta *cu sedilă*, în caz că este necesară pentru compatibilitate, se poate activa folosind aceeași comandă dar adăugând și parametrul `cedilla`.

13.5 Configurarea sistemului din KDE

În Kubuntu, toate configurările importante ale sistemului sunt reunite într-un panou de comandă numit **System Settings**. Acest panou este asemănător ca funcționalitate cu **Control Panel** din Windows și poate fi accesat din meniul de KDE (aflat în stânga jos, în mod implicit).

13.5.1 System Settings

Pentru Kubuntu 9.10, fereastra de configurare este prezentată în figura 13.9.

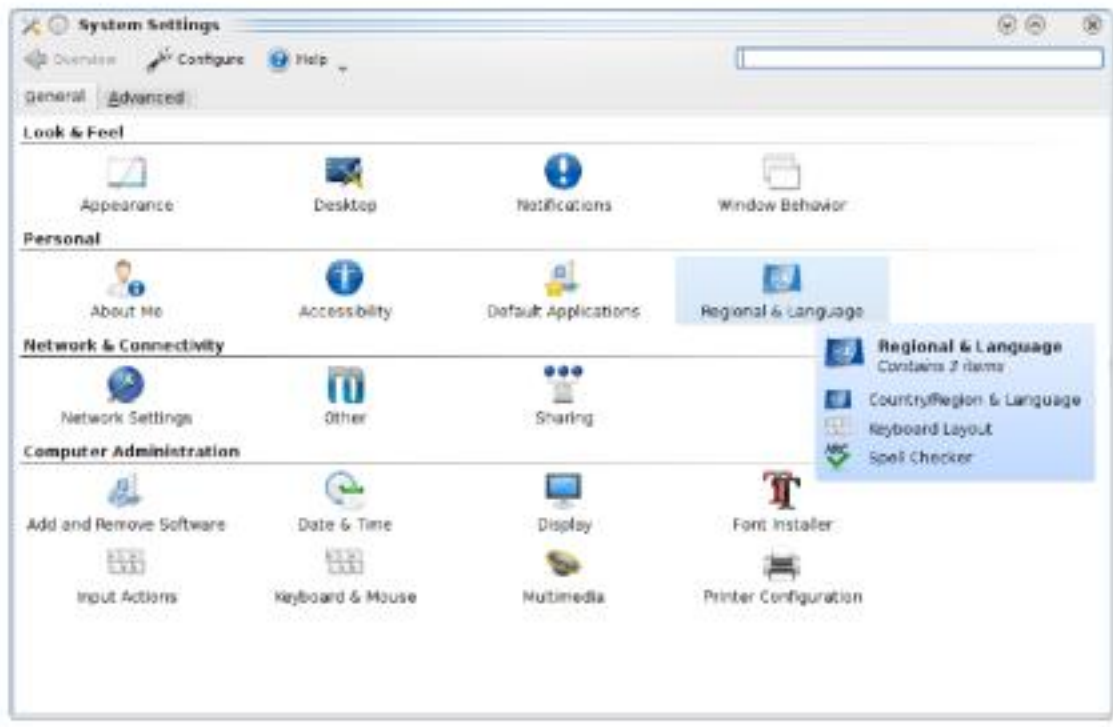


Figura 13.9: System Settings în Kubuntu 9.10

Ea oferă acces la principalele configurări desktop-ului și ale sistemului de operare: configurări legate de aspectul vizual, configurarea perifericelor, configurarea rețelei etc.

13.5.2 Schimbarea aspectului interfeței grafice

Appearance conține configurări legate de modul în care arată interfața grafică. Se poate alege schema de culori a ferestrelor și a butoanelor (*Colors*), fonturile folosite pentru afișarea textelor din interfața grafică (*Fonts*), icoanele folosite în sistem (*Icons*), stilul în care arată elementele vizuale(*Style*), felul în care arată bara de titlu a ferestrelor (*Window Decorations*), animația care arată stadiul curent al procesului de înărcare KDE (*Splash Screen*). Figura 13.10 prezintă un exemplu de configurări vizuale.

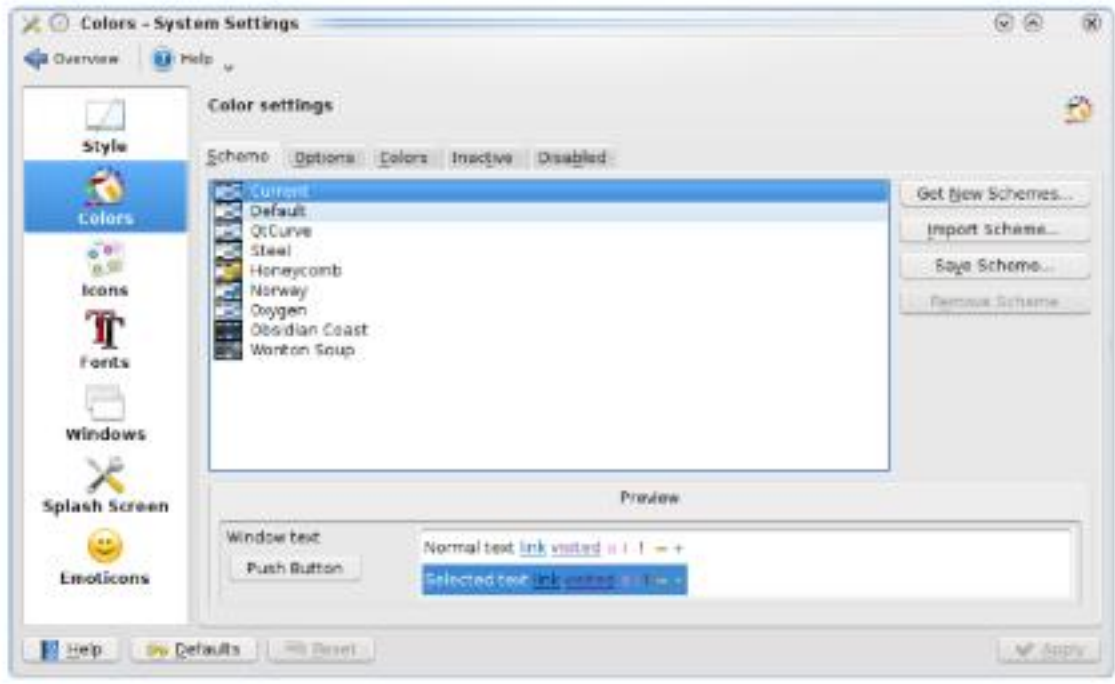


Figura 13.10: Alegerea schemei de culori

Desktop permite modificarea fundalului (*Background*), a screen saver-ului (*Screen Saver*) sau a numărului de desktop-uri (*Multiple Desktops*). În mod normal în Linux se folosesc două sau patru desktop-uri. Fiecare poate avea propria imagine de fundal și propriile programe deschise.

În general, aplicațiile sunt grupate de către utilizatori pe desktop-uri în funcție de utilitate. De exemplu, pe desktop-ul 1 se poate deschide un web browser, pe desktop-ul 2 un client de e-mail, pe desktop-ul 3 o temă de casă și pe desktop-ul 4 un program de mesagerie instant. Avantajul oferit de această abordare este acela că fiecare desktop devine mai liber, având mai puține programe deschise. De asemenea, acest mod de organizare permite mai ușor concentrarea atenției pe anumite activități.

Notifications conține configurări legate de sunetele asociate diferitelor acțiuni din sistemul de operare (*System Notifications*). De asemenea tot în această secțiune se pot defini și aplicațiile care pornesc atunci când un disc este inserat în unitatea optică (echivalentul opțiunii autorun din Windows) (*Storage Media Notifications*).

13.5.3 Configurări de bază ale sistemului de operare

About Me permite introducerea datelor personale ale utilizatorului și specificarea căii către directoarele sale personale: directorul *home*, directorul unde se află fișierele de pe *Desktop* și directorul `Autostart`. În directorul de `Autostart` se pot pune scripturi care vor fi rulate la pornirea KDE.

Accesibility oferă acces la diferire configurări pentru persoanele cu dizabilități.

Default Applications permite specificarea aplicațiilor folosite la deschiderea diferitelor tipuri de fișiere.

Regional & Language conține configurări legate de regiunea în care se găsește utilizatorul (de exemplu modul în care sunt afișate numerele, separatorul pentru partea zecimală a numerelor, simbolul pentru moneda utilizată, tipul de calendar folosit, formatul orei și al datei). Tot în această secțiune se poate alege tipul de tastatură (de exemplu cu 104 sau 105 taste) și aranjamentul caracterelor pe taste. Figura 13.11

prezintă configurația pentru tastatura în limba română.

Figura 13.11: Tastatura în limba română

13.5.4 Configurări administrative

Date & Time permite configurarea datei și a orei sistemului. Tot aici se poate alege și fusul orar corespunzător zonei în care se găsește utilizatorul.

Keyboard & Mouse conține configurări legate de tastatură și mouse. Una din cele mai importante opțiuni prezente aici este cea legată de starea `NumLock` la inițializarea KDE. Se poate alege ca `NumLock` să fie activat, dezactivat sau să își păstreze starea anterioară. În cadrul subsecțiunii *Mouse* se poate alege tema cursorului de la mouse.

Tot în această secțiune se pot configura combinațiile de taste (*shortcut*) pentru diferite funcții din KDE. Cele mai utilizate combinații (cu valorile lor implicite) sunt prezentate în tabelul 13.1.

Display permite, printre altele, alegerea uneia dintre rezoluțiile de ecran disponibile și a parametrilor de culoare (gamma, luminozitate, contrast).

Multimedia conține configurări ale plăcii audio.

13.5.5 Configurarea rețelei

În *System Settings*, la **Network Settings** se pot face configurări ale rețelei.

În *Network Connections* > *Wired* apare lista conexiunilor configurate (figura 13.12). Alegând una din aceste conexiuni se pot configura parametrii ei [referință la parametrii de rețea], precum în figura 13.13.

13.5.6 Managementul utilizatorilor

Configurarea utilizatorilor se realizează folosind aplicația KDE User Manager (**`kuser`**) – figura 13.14.

Tabelul 13.1: Combinații de taste uzuale în KDE

Combinația de taste	**Acțiune**
`F1`	Deschide fereastra de ajutor
`Alt+F2`	Deschide fereastra de Run de unde se poate porni o aplicație specificând numele executabilului
`Ctrl+Esc`	Pornește aplicația *System Activity* în care este afișată o listă a proceselor active
`Ctrl+Fn`	Face trecerea la desktop-ul `n` (`n` este un număr între 1 și 4)
`Alt+Tab` sau `Alt+Shift+Tab`	Face trecerea de la o aplicație la alta. Folosind tasta `Shift` se ciclează aplicațiile în ordine inversă
`Alt+F4`	Închide fereastra activă
`Ctrl+Alt+Esc`	Această combinație permite terminarea unei aplicații în mod forțat. După ce combinația de taste a fost efectuată, cursorul mouse-ului se va transforma (cel mai uzual într-un `X`) și fereastra pe care se va realiza click va fi terminată în mod forțat. Funcționarea aplicației este similară comenzii **`kill`**

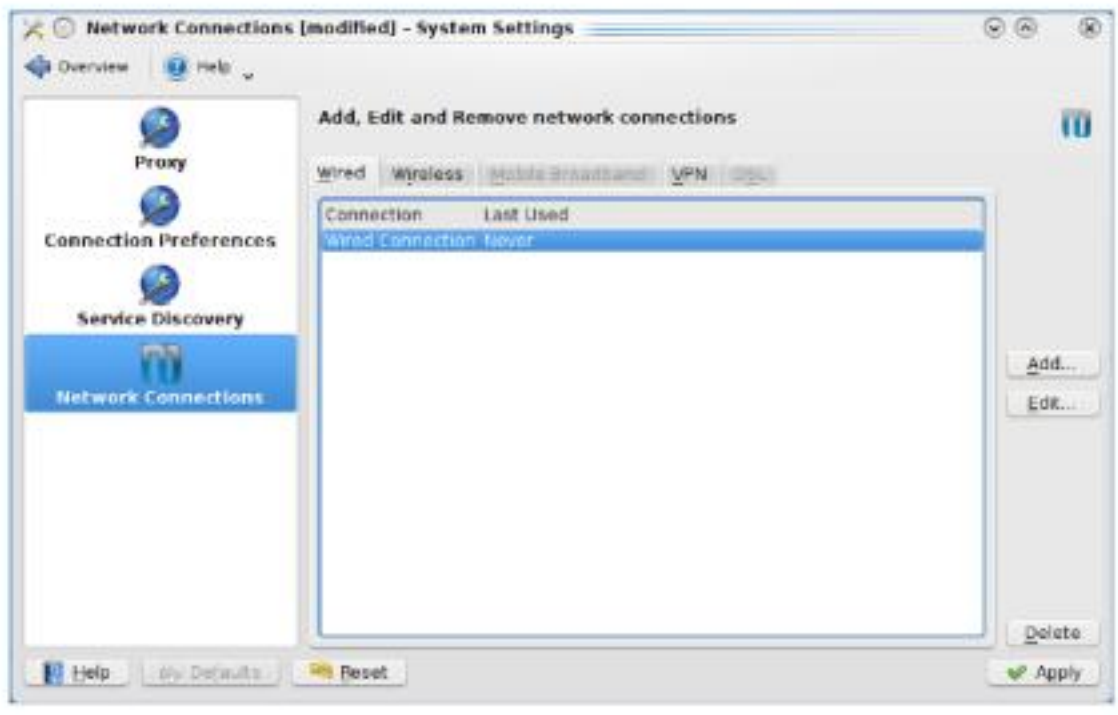


Figura 13.12: Configurarea conexiunilor la rețea

Aici se pot adăuga, șterge sau modifica utilizatorii din sistem. Se pot modifica numele de login, numele real al utilizatorului, grupurile din care face parte etc. Pentru a modifica datele unui utilizator se selectează utilizatorul dorit și se apasă butonul *Modify*, fereastra afișată fiind prezentată în figura 13.15.

13.5.7 Monitorizarea sistemului și managementul proceselor

Aplicația KDE pentru monitorizarea sistemului este **System Monitor** (comanda **`ksysguard`**). Aceasta vine implicit împreună cu Kubuntu.

Aplicația are două tab-uri în configurația implicită: unul pentru managementul și monitorizarea proceselor (figura 13.16) și unul pentru monitorizarea sistemului

Figura 13.13: Configurarea parametrilor de rețea de bază

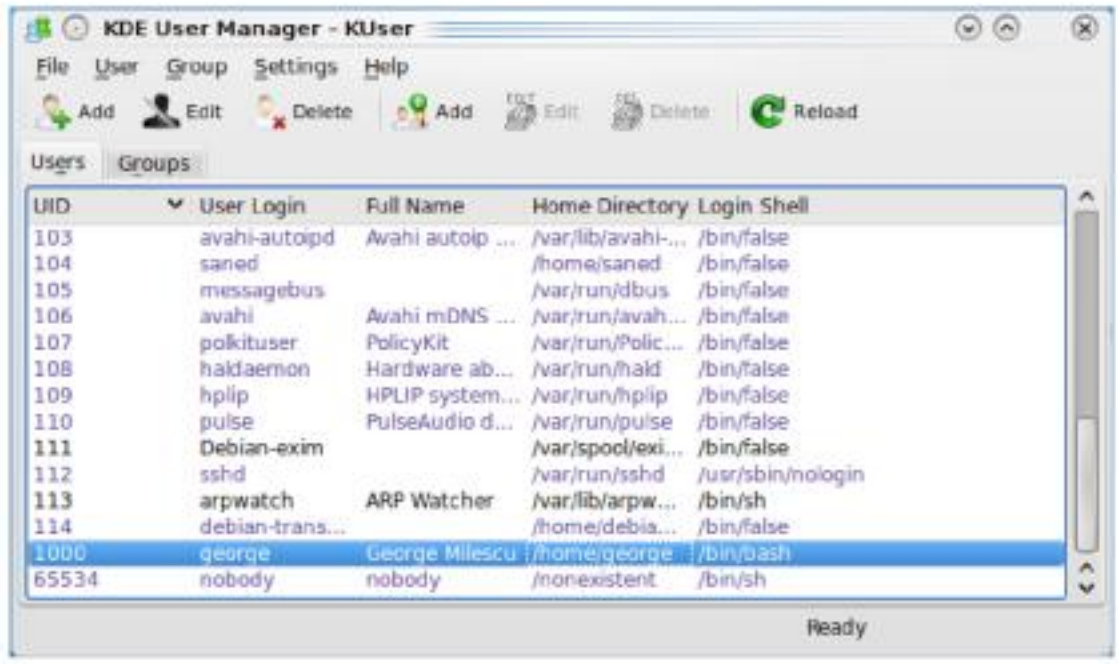


Figura 13.14: Administararea utilizatorilor în KDE User Manager

(figura 13.17). Dacă se dorește, lista de procese poate fi afișată sub formă de arbore.

De asemenea, în partea de sus a ferestrei se află un câmp *Search* se permite căutarea unui proces în listă după numele lui. Un proces poate fi oprit în mod forțat dacă este selectat din listă și este apăsat butonul *Kill*.

Pentru monitorizarea sistemului, aplicația pune la dispoziție o colecție de senzori ale căror valori pot fi grupate în tab-uri personalizate, precum se poate observa în figura 13.18.

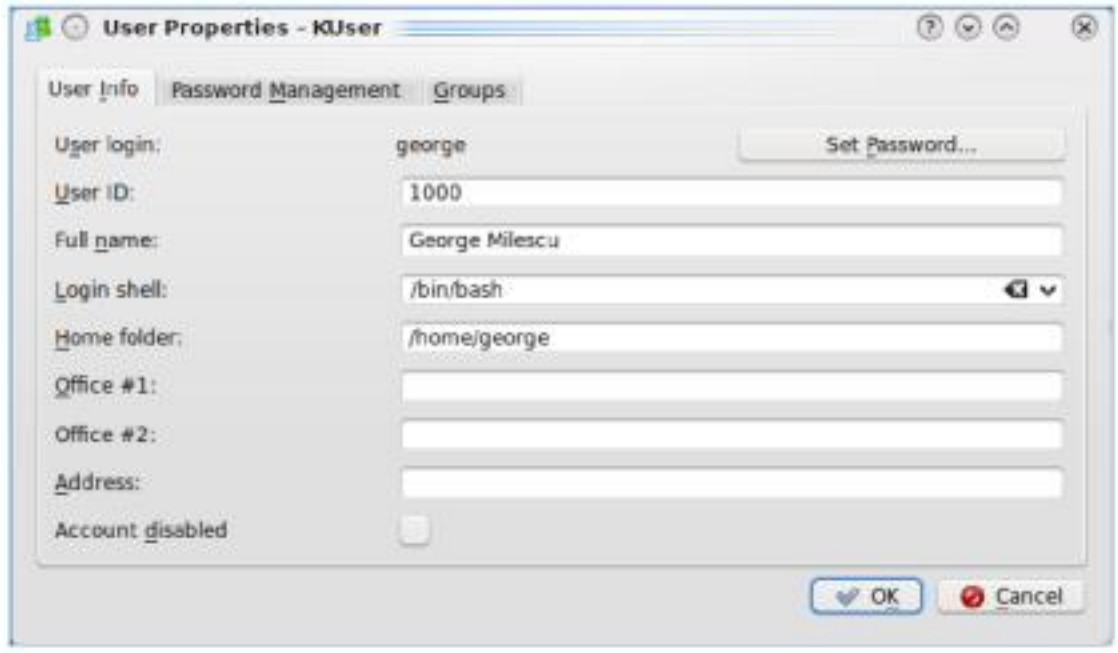


Figura 13.15: Modificarea datelor unui utilizator

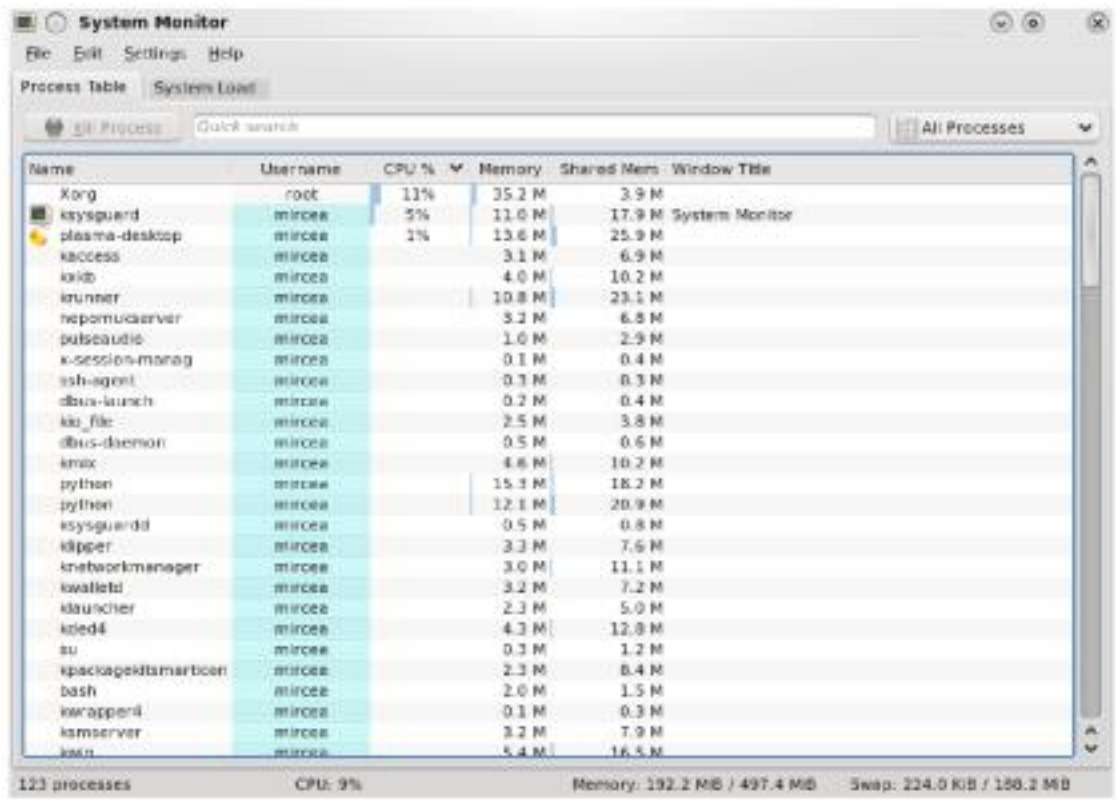


Figura 13.16: System Monitor – lista de procese

13.5.8 Gestiunea pachetelor

Există mai multe utilitare pentru a realiza gestiunea pachetelor folosind interfața grafică. În Kubuntu, aplicația utilizată pentru managementul pachetelor se numește *KPackageKit*. Figura 13.19 prezintă interfața *KPackageKit* pentru realizarea actualizărilor.

Aplicația permite gestiunea repository-urilor, actualizarea bazei de date cu pachete, căutarea de pachete după nume, instalarea de pachete noi, dezinstalarea de pachete și actualizarea tuturor pachetelor din sistem.

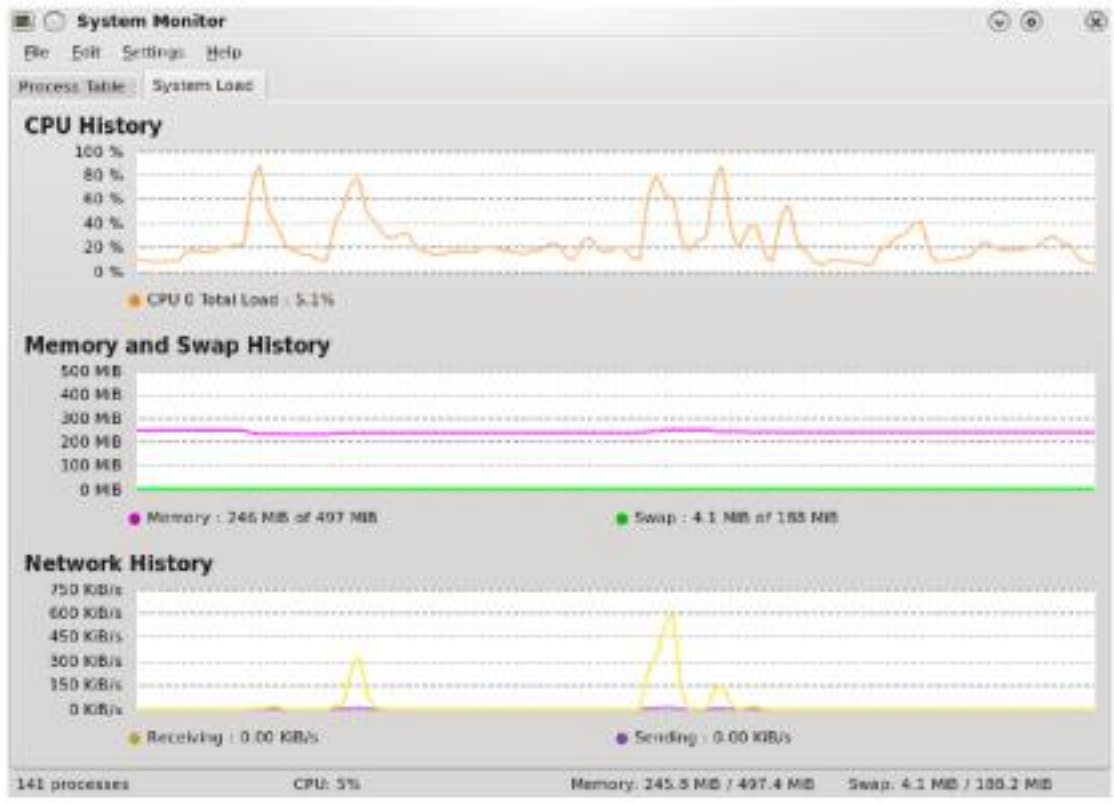


Figura 13.17: System Monitor – monitorizarea sistemului

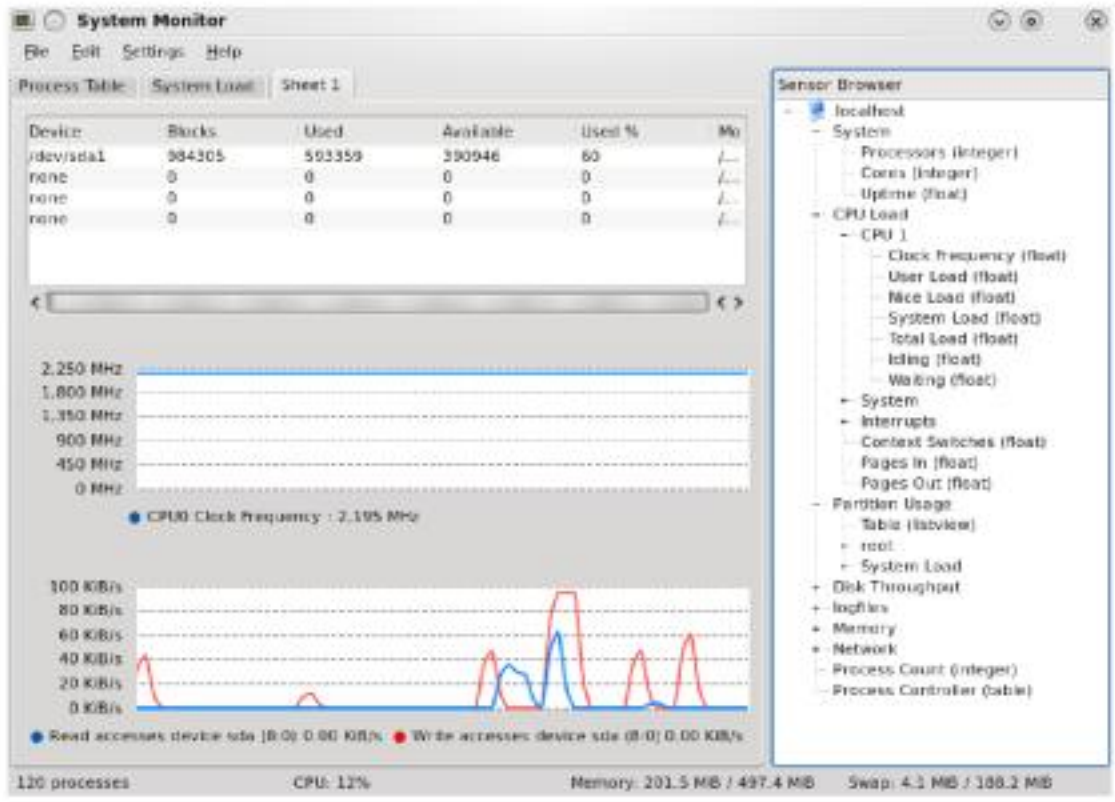


Figura 13.18: System Monitor – senzori care pot fi folosiți în tab-uri personalizate

13.6 Servicii desktop

13.6.1 Pornirea facilă a aplicațiilor

KDE versiunea 4 a introdus în KDE un nou serviciu de pornire a aplicațiilor. El înlocuiește vechea fereastră de Run (asemănătoare cu cea din Windows) cu o fereastră în care utilizatorul este ajutat dinamic să identifice aplicația pe care dorește să o pornească. Accesul la aceasta fereastra de execuție se face folosind combinația de taste `Alt+F2` – aceeași combinație ca în KDE3 și în Gnome.

În urma introducerii câtorva litere din numele sau descrierea aplicației (figura 13.20), fereastra se va completa cu aplicațiile posibile. Aceeași fereastră poate fi folosită și

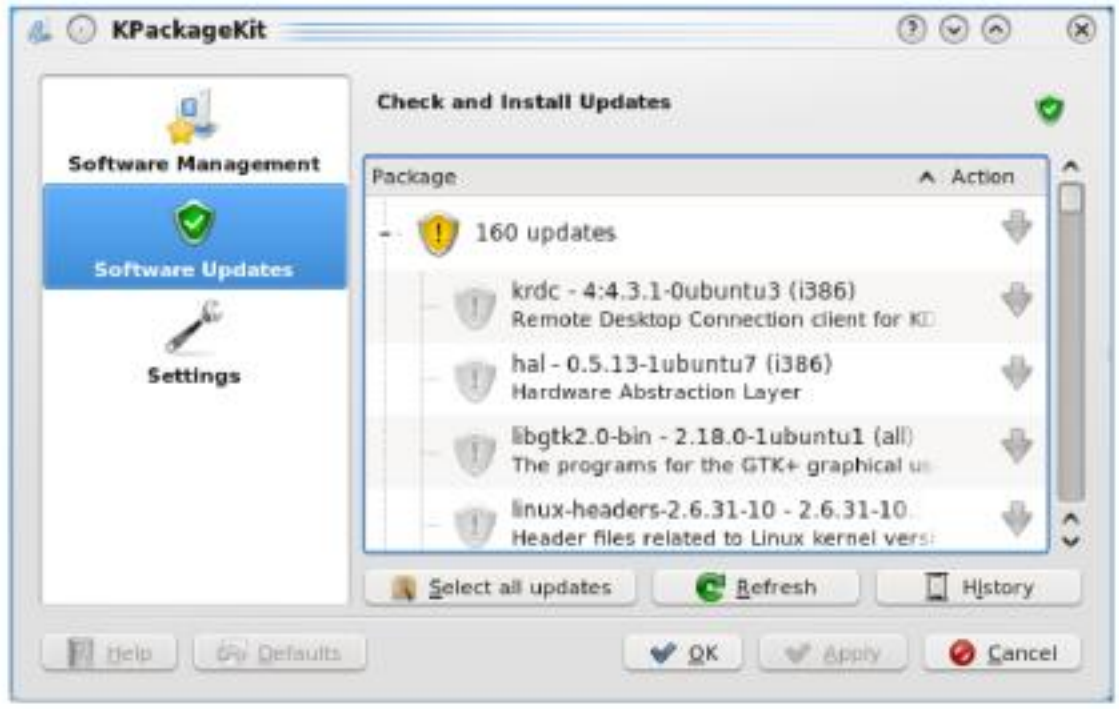


Figura 13.19: KPackageKit – aplicația pentru managementul pachetelor

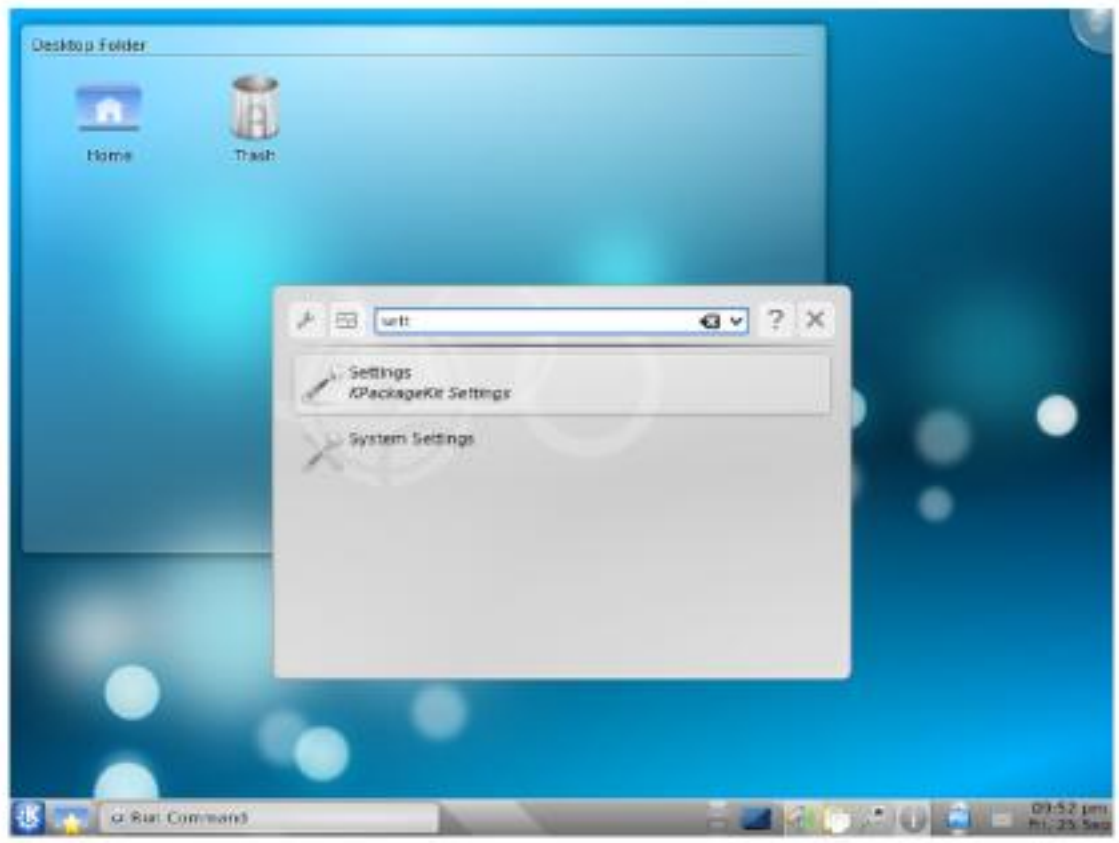


Figura 13.20: Fereastra de execuție a aplicațiilor

pentru realizarea de calcule simple – doar se introduce calculul dorit urmat de semnul egal (spre exemplu: `.19 * 2000 =`) și rezultatul apare în listă.

13.6.2 Căutarea fișierelor în sistem

În prezent, pentru ca utilizatorul să poată găsi date pe calculatorul propriu, fiecare sistem de operare modern oferă un serviciu de căutare a fișierelor. În general, acest serviciu se bazează pe un daemon care indexează permanent fișierele din sistem. La indexare, pe lângă numele și locația fișierelor, daemon-ul de indexare poate colecta atât conținut cât și metadate ale fișierului (spre exemplu, în cazul fișierelor `MP3`, se indexează și tag-urile fișierului precum artistul, numele melodiei etc.).

Serviciile de căutare din sistemele de operare pe cele mai cunoscute sunt următoarele:

- Windows – *Windows Search*[1]
- Linux – Strigi[2] (KDE), Beagle[3] (GNOME)
- Mac OS X – *Spotlight*[4]

Utilizatorul poate instala și soluții *third-party* pentru realizarea căutărilor în sistem. Cea mai cunoscută soluție pentru cautare pe desktop oferită ca aplicație separată este *Google Desktop Search*[5].

13.6.3 Notificări

Sistemele de operare moderne pun la dispoziția aplicațiilor servicii de notificare a utilizatorului. Astfel, notificările au un aspect consistent, efortul pentru dezvoltarea aplicațiilor scade și crește nivelul de utilizabilitate. În Linux există suport pentru notificări generice prin intermediul bibliotecii `libnotify`.

Pe lângă sistemele de notificare puse la dispoziție de desktop, majoritatea interfețelor grafice pun la dispoziție o zonă în dreptul ceasului pentru icoanelor aplicațiilor care doresc să notifice utilizatorul (pe bara de aplicații sau pe bara cu meniul desktop-ului). Această zonă mai este numită *tray* (sau *system tray*).

13.7 Aplicații KDE vs. GNOME

Alegerea între KDE și GNOME aparține fiecărui utilizator și se poate baza pe mai multe criterii, precum aspectul vizual și nivelul de funcționalitate dorit. Tabelul 13.2 prezintă comparativ aplicațiile cele mai uzuale de KDE și GNOME.

Aplicațiile prezentate nu reprezintă restricții de utilizare. Ele se pot executa atât pe și KDE cât și pe GNOME dacă sunt instalate bibliotecile necesare[6].

13.8 Studii de caz

13.8.1 Configurarea X peste SSH

Așa cum a fost prezentat și la începutul capitolului, arhitectura **X Window System** permite comunicarea prin rețea între clienții X și serverul X. Această comunicare se poate realiza folosind mai multe protocoale, printre care și SSH. SSH este un protocol care permite comunicarea între două calculatoare utilizând un canal securizat (secțiunea 9.2.2). SSH poate astfel securiza mesajele dintre clienții și serverul X

[1] http://www.microsoft.com/windows/products/winfamily/desktopsearch/default.mspx
[2] http://strigi.sourceforge.net/
[3] http://beagle-project.org/
[4] http://www.apple.com/macosx/what-is-macosx/spotlight.html
[5] http://desktop.google.com/
[6] bibliotecile sunt instalate automat la instalarea aplicațiilor

Tabelul 13.2: Aplicații KDE și GNOME

Aplicație	KDE	GNOME
Configurarea Sistemului	System Setings	Meniul System
Gestiunea pachetelor	KPackageKit	Synaptic Packet Manger
Monitorizarea sistemului	System Monitor (KSysGuard)	System Monitor
File Manager	Konqueror, Dolphin	Nautilus
Editor de text	Kate	Gedit
Emulator de consolă	Konsole	gnome-terminal
Calculator	Kcalc	Gcalctool
Video Player	Kaffeine	Totem
Audio Player	Amarok	Rhythmbox
Înscriptor de CD/DVD	K3B	Brasero
Viewer de imagini	Gwenview	Eye of GNOME
Viewer PDF	Okular	Evince
Browser Web	Konqueror	Epiphany

(figura 13.21). Opțiunea SSH care permite acest lucru poate se numește *X11 Forwarding*.

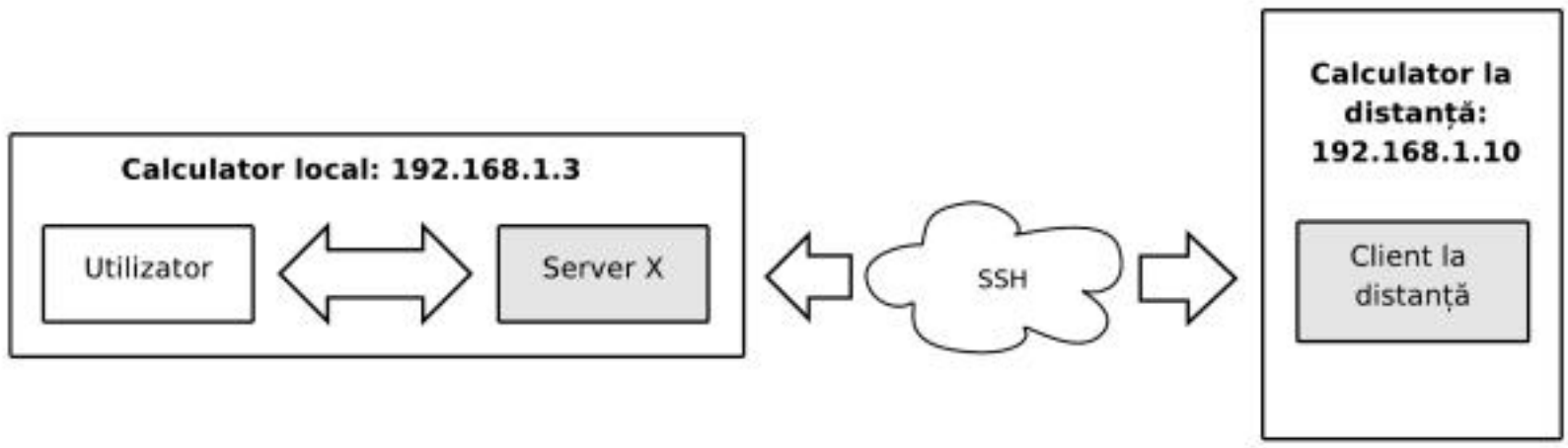


Figura 13.21: Conexiunea client – server X peste SSH

Pentru a putea rula un client X aflat la distanță peste SSH, primul pas este configurarea protocolului SSH în acest sens (activarea *X11 Forwarding*). Cea mai simplă metodă este specificarea parametrului `-X` atunci când se realizează conexiunea SSH de pe calculatorul utilizatorului (unde este server-ul X) pe calculatorul unde se va executa clientul X.

Comanda cu care se realizează conexiunea SSH pentru a permite încapsularea protocolului X este:

```
1 george@asgard:~$ ssh george@192.168.1.10 -X
```

După ce conexiunea SSH a fost stabilită, se poate porni un client X. De exemplu, se poate porni *KCalc*:

```
1 george@acasa:~$ kcalc
```

După cum se poate observa în figura 13.22, bara de titlu a ferestrei *KCalc* specifică faptul că această fereastră corespunde unui client care nu se găsește pe calculatorul local (`<@acasa>`).

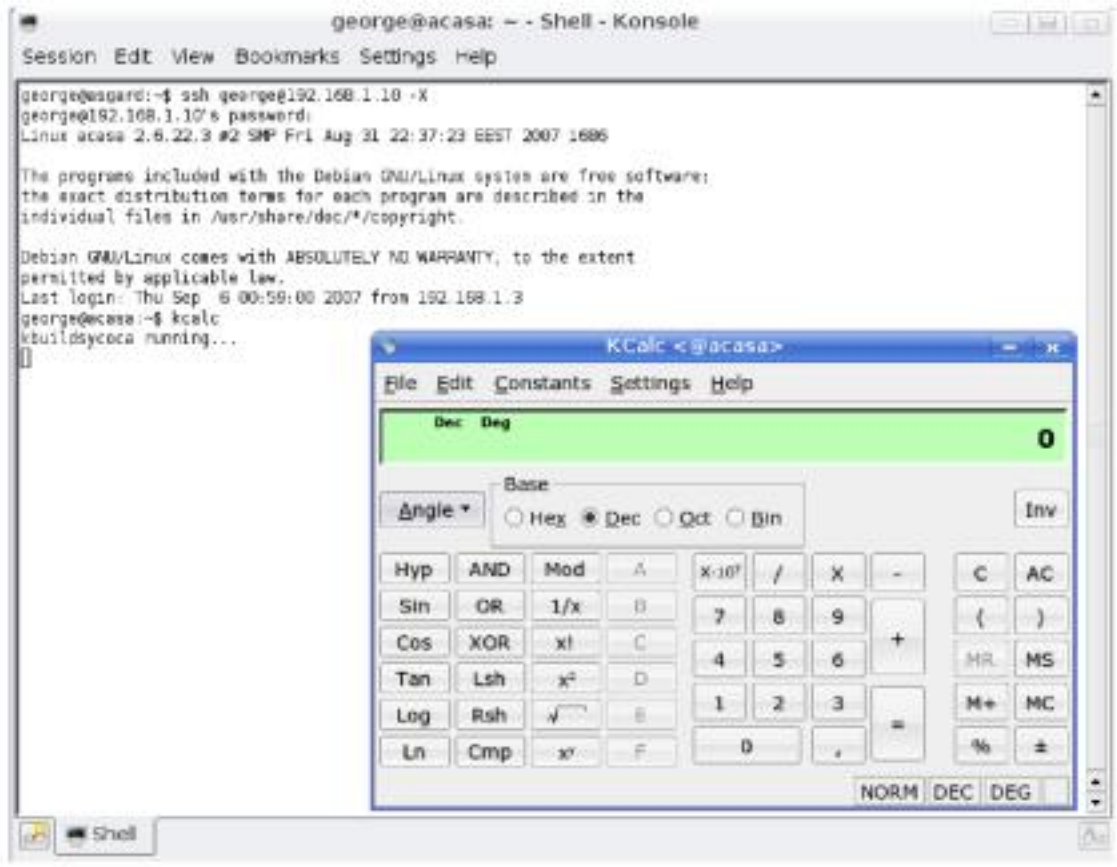


Figura 13.22: Pornirea unei aplicații X peste SSH

Folosind X peste SSH se poate executa și un întreg *Desktop Environment*, nu doar programe individuale. Acest lucru nu este însă uzual datorită alternativelor mai bune existente în acest sens (**vnc**, **freenx** etc.). Conexiunea SSH necesită criptare și decriptare – acest lucru presupune consumul unor resurse hardware suplimentare.

Pentru a putea rula X peste SSH folosind pe calculatorul local Windows (server X) și Linux pentru clienții X, trebuie ca pe calculatorul local să fie instalat *Cygwin* (sau un alt server X instalabil pe Windows).

13.8.2 Instalarea și configurarea VNC

VNC (*Virtual Network Computing*) este unul dintre cele mai folosite sisteme de vizualizat desktop-ul unui calculator aflat la distanță. VNC se bazează pe protocolul *RFB* (*Remote Frame Bufer*) și de aceea poate fi folosit între sisteme de operare diferite (Windows, Linux, Mac OS X).

VNC funcționează pe sistemul client – server. Pe calculatorul aflat la distanță (al cărui desktop va fi vizualizat) rulează un server de VNC la care se conectează un client aflat pe calculatorul local. La un server se pot conecta mai mulți clienți simultan. Un client poate să vizualizeze desktop-ul calculatorului pe care rulează serverul VNC sau, dacă este permis din configurarea server-ului, poate să și interacționeze cu el (de exemplu schimbând poziția cursorului mouse-ului). Figura 13.23 prezintă o astfel de sesiune VNC.

Serverul de VNC preia de câteva ori pe secundă imaginea care se afișează pe ecran și o trimite tuturor clienților. În cazul în care clienții îi cer acest lucru, serverul poate accepta de la aceștia cereri de mișcare a cursorului mouse-ului, de efectuare de click-uri de mouse sau de introducere de caractere. În acest fel, utilizatorul poate prelua controlul calculatorului pe care rulează serverul de VNC.

Figura 13.23: Conectarea prin VNC folosind un client Windows și un server Linux

Folosind VNC este posibil ca mai mulți utilizatori să aibă simultan controlul unui calculator. De cele mai multe ori, înainte de a putea accesa un server de VNC, utilizatorul trebuie să se autentifice. Acest lucru se realizează pe baza unui nume de utilizator și a unei parole.

Atât în Windows cât și în Linux există mai multe programe, atât server cât și client, care implementează VNC. Pentru Linux, unul dintre cele mai simple servere VNC este **`x11vnc`**. Odată instalat, acest server este pornit prin apelarea comenzii **`x11vnc`**:

```
1  george@acasa:~$ x11vnc -usepw
2  10/09/2007 22:48:59 -usepw: found /home/george/.vnc/passwd
3
4  [...]
5  10/09/2007 22:48:59 *** Trying ":0" in 4 seconds.  Press Ctrl-C to abort.
6
7  10/09/2007 22:48:59 *** 1 2 3 4
8  10/09/2007 22:49:03 *** XOpenDisplay of ":0" successful.
9
10 10/09/2007 22:49:03
11 10/09/2007 22:49:03 Using X display :0
12 10/09/2007 22:49:03
13 10/09/2007 22:49:03 X display :0.0 is 32bpp depth=24 true color
14
15 FrameBuffer Info:
16  width:              1280
17  height:             1024
18  scaled_width:       1280
19  scaled_height:      1024
20
21 [...]
22
23 The VNC desktop is:      acasa:0
24 PORT=5900
```

Comanda trebuie apelată de către utilizatorul care este autentificat în interfața grafică. Implicit, după ce ultimul client se deconectează, serverul se va opri.

Pentru a securiza accesul clienților pe baza unei parole, **`x11vnc`** trebuie pornit cu opțiunea `-usepw`. În acest caz serverul va verifica dacă există fișierul

`~/.vnc/passwd`. În cazul în care fișierul există, din el va fi citită parola ce va fi cerută clienților. Dacă fișierul nu există, utilizatorului i se va cere o parolă ce va fi stocată în `~/.vnc/passwd` și va fi folosită la următoarele instanțe ale serverului.

Există mai mulți clienți VNC în Linux: **`xvnc4viewer`**, **`xvncviewer`** etc.

Şi în Windows există servere cât clienți VNC. Unele programe sunt gratuite (ca de exemplu **tightVNC**), altele trebuie cumpărate (ca de exemplu **RealVNC**, ce dispune și de o versiune gratuită dar limitată ca opțiuni).

O alternativă la VNC este **FreeNX**. Acest program trimite datele între client și server peste SSH, însă comprimă datele înainte de a le trimite. În acest fel reușește să obțină performanțe superioare VNC-ului ca timp de răspuns și ca lățime de bandă consumată.

13.8.3 Remote Desktop Connection

RDC (*Remote Desktop Connection*) este o aplicație care funcționează în mod similar cu VNC și care permite unui utilizator să acceseze de la distanță desktop-ul unui calculator. Această aplicație este disponibilă doar împreună cu *Windows XP Professional* sau cu *Windows Vista Business*, *Ultimate* sau *Enterprise*.

În comparație cu VNC, RDC are avantajul că este instalat odată cu instalarea Windows. Dezavantajul RDC este acela că nu permite accesarea desktop-ului simultan local și de la distanță. Cu alte cuvinte, atunci când un utilizator este autentificat local pe calculator și interacționează direct cu sistemul de operare, nu este posibil ca altcineva să fie autentificat prin RDC și să vadă acțiunile primului utilizator. În momentul în care se încearcă autentificarea prin RDC, utilizatorul local este deconectat. Atunci când un utilizator aflat la distanță este autentificat prin RDC și un utilizator local încearcă și el să se autentifice, utilizatorul aflat la distanță este deconectat.

Pentru utilizarea RDC, acesta trebuie activat pe calculatorul *server* (calculatorul care va fi controlat de la distanță). Pentru a realiza acest lucru pe Windows XP se accesează *System Properties* (click dreapta pe *My Computer*, apoi *Properties*), după care din tabul *Remote* se bifează *Allow users to connect remotely to this computer* (figura 13.24).

Este posibil să fie necesare configurări ale programelor de tip firewall (vezi secțiunea 10.5.1) existente pe cele două calculatoare între care se dorește realizarea legăturii.

Pentru a realiza conexiunea, de pe calculatorul local trebuie pornit clientul de RDC (figura 13.25). Acesta se accesează prin *Start* > *All Programs* > *Accessories* > *Communications* > *Remote Desktop Connection* sau *Start* > *All Programs* > *Accessories* > *Remote Desktop Connection*.

Pentru conectarea la calculatorul aflat la distanță este necesară adresa IP a acestuia. Folosind butonul *Options* se pot configura parametrii conexiunii RDC (figura 13.26).

După ce conexiunea a fost inițializată, utilizatorul trebuie să se autentifice (figura 13.27).

După ce autentificare este realizată cu succes, desktop-ul calculatorului afla la distanță poate fi accesat, asemănător unei conexiuni VNC.

Figura 13.24: Activarea RDC

Figura 13.25: Fereastra clientului de RDC

Cuvinte cheie

- aplicații grafice
- client X
- desktop environment: KDE, GNOME
- diacritice
- display manager: **`gdm`**, **`xdm`**, **`kdm`**
- font
- grafică raster
- grafică vectorială
- GUI
- keyboard layout
- protocolul X
- rezoluție
- RDC
- server X
- servicii desktop
- Unicode
- UTF-8
- VNC
- window manager: Kwin, Metacity, Compiz
- X Forwarding
- X Window System
- X.Org
- **`xinit`**, **`startx`**
- Xrandr

Figura 13.26: Configurarea opțiunilor conexiunii RDC

Figura 13.27: Autentificarea RDC

Întrebări

1. Care din următoarele formate de fișiere NU este un format vectorial?
   - ❑ PDF
   - ❑ True Type font
   - ❑ PNG
   - ❑ SVG
2. Care dintre următoarele este un standard pentru descrierea modul de codificare a caracterelor intr-un fișier?

- ❑ True Type
- ❑ Unicode
- ❑ UTF-8
- ❑ RGB

3. Un Display Manager are rolul de a:
   - ❑ controla felul în care arată ferestrele
   - ❑ permite unui utilizator să se autentifice în interfața grafcă
   - ❑ controla deschiderea și închiderea ferestrelor
   - ❑ gestiona conținutul ferestrelor
4. Care dintre următoarele perechi NU conține elemente cu același rol?
   - ❑ KDE și GNOME
   - ❑ **kdm** și **gdm**
   - ❑ Kwin și Metacity
   - ❑ Xorg și bitmap
5. Utilizatorul Dorel lucrază pe un calculator cu interfață grafică instalată, dar care NU este pornită. El încearcă să pornească interfața folosind comanda următoare dar nu reușește. De ce?

```
1 dorel@home:~$ service kdm start
```

   - ❑ scriptul `kdm` nu porneste interfața grafică
   - ❑ Dorel nu are dreptul să pornească serviciul `kdm`
   - ❑ Dorel trebuia să ruleze scriptul **`startx`** înainte de a rula scriptul `kdm`
   - ❑ Dorel trebuia să ruleze scriptul **`startx`** după ce a rulat scriptul `kdm`
6. Care dintre urmatoarele NU este o metoda de control de la distanță a unui calculator?
   - ❑ X peste SSH
   - ❑ VNC
   - ❑ Spotlight
   - ❑ Remote Desktop Connection
7. Care dintre următoarele NU este un Window Manager?
   - ❑ KDE
   - ❑ Compiz
   - ❑ Kwin
   - ❑ Metacity

8. Care dintre următoarele aplicații este utilizată în mediul grafic pentru monitorizarea ocupării resurselor hardware?
   - ❑ top
   - ❑ System Information (**`ksysguard`**)
   - ❑ htop
   - ❑ free
9. Un Window Manager controlează felul în care arată ferestrele aplicațiilor. Un client X preia intrărea (input-ul) direct de la utilizator.
   - ❑ adevărat, adevărat
   - ❑ adevărat, fals
   - ❑ fals, adevărat
   - ❑ fals, fals
10. Care dintre următoarele NU reprezintă un client X?
    - ❑ Kcalc
    - ❑ gnome-terminal
    - ❑ Xorg
    - ❑ kdm

Capitolul 14

Utilitare pentru dezvoltare

First learn computer science and all the theory. Next develop a programming style. Then forget all that and just hack.

George Carrette

Ce se învață din acest capitol?

- Convenții de codare
- Editorul Vim
- Sisteme de control a versiunii; Subversion și Git
- Parcurgerea rapidă a codului
- Compilarea unei aplicații din surse
- Zonele unui executabil și proces
- Depanarea unui program folosind GDB, ddd și Valgrind
- IDE-uri: Eclipse, Anjuta, Microsoft Visual Studio
- Gestiunea unui proiect software

14.1 Introducere

Dezvoltarea aplicațiilor este una din principalele activități ale specialiștilor în calculatoare. Procesul de dezvoltare necesită un set de utilitare pe care programatorul (dezvoltatorul software le folosește), de la editoare și utilitare de parcurgere a codului, până la utilitare de depanare și investigare a execuției aplicației. Fără a își propune să acopere la nivel de detaliu aceste utilitare, capitolul de față prezintă principalele utilitare folosite de un dezvoltator de software pe un sistem Unix/Linux. Multe dintre acestea sunt portate pe Windows și Mac OS X. În general, dezvoltarea de aplicații pe un sistem Linux presupune folosirea de utilitare dedicate (editoare, compilatoare, depanatoare, mecanisme de documentare), în vreme ce dezvoltatorii de aplicații pe sisteme Windows folosesc, de obicei, un mediu integrat de dezvoltare (IDE – *Integrated Development*

Environment); cel mai cunoscut este Microsoft Visual Studio (vezi secțiunea 14.12.1). Totuși, și în Linux sunt folosite IDE-uri precum Eclipse sau Anjuta (vezi secțiunea 14.9).

14.2 Coding style (indent, astyle)

Una dintre greșelile frecvente realizate de programatorii începători este credința că vor putea înțelege codul sursă scris de ei înșiși oricând în viitor[1]. Scrierea unui cod dezorganizat, fără comentarii este, de obicei, echivalentă cu scrierea unui cod nementenabil.

Pentru a preveni situațiile în care se consumă timp pentru înțelegerea codului scris de altcineva trebuie respectate un set de norme de bază pentru editarea codului. Folosirea acestor norme ajută și lucrul într-o echipă de dezvoltare, când mai mulți dezvoltatori ajung să parcurgă codul scris de un altul.

Diverse proiecte software colectează normele preferate (sau obligatorii) de redactare a codului într-un document de stil de codare (*coding style*, *programming style*, *coding standards*, *code conventions*). Exemple sunt Java[2], GNU[3], BSD[4], Linux[5], Python[6], PHP[7] etc.

Stilurile de codare variază în funcție de limbajul de programare sau de proiect, dar există un set de reguli comune care facilitează înțelegerea și îmbunătățirea ulterioară a codului. Exemple de astfel de reguli sunt:

- folosirea de nume relevante pentru variabile; se preferă denumirea `num_files` în loc de `num_var`, `EncryptMessage` în loc de `DoStuff` sau `tmp` în loc de `TemporaryVariabileForStoringIntegerArraySize`; variabilele de genul `i`, `j` se consideră implicite pentru parcurgerea vectorilor iar folosirea lor nu trebuie detaliată;
- indentarea codului; corpul unui bloc de instrucțiuni se indentează fie cu TAB fie cu spații;
- spații/linii libere; pentru lizibilitate se recomandă folosirea spațiilor libere între argumentele unei funcții, înainte și după simbolul = în momentul inițializării unei variabile etc. și a liniilor libere pentru separarea blocurilor de instrucțiuni;
- funcții kilometrice; corpul funcțiilor trebuie limitat pentru a fi ușor de parcurs; funcții care depășesc două ecrane trebuie evitate;
- comentarii; codul trebuie comentat pentru a facilita înțelegerea acestuia; comentariile nu trebuie să fie redundante, trebuie să ajute programatorul acolo unde codul este dificil de înțeles;

[1] Eagleson's Law of Programming: Any code of your own that you haven't looked at for six or more months, might as well have been written by someone else.

[2] http://java.sun.com/docs/codeconv/

[3] http://www.gnu.org/prep/standards/

[4] http://www.freebsd.org/cgi/man.cgi?query=style&sektion=9

[5] http://lxr.linux.no/#linux+v2.6.31/Documentation/CodingStyle

[6] http://www.python.org/dev/peps/pep-0008/

[7] http://pear.php.net/manual/en/standards.php

- consecvența stilului; este mai bine să se scrie un cod greu lizibil, dar care are o consecvență în stil – chiar ilizibil – decât să se folosească mai multe stiluri, fie ele și bune.

indent, astyle În situațiile în care fișierele sursă conțin cod indentat necorespunzător, se pot folosi utilitare precum `indent` sau `astyle`.

Utilitarul `indent` indentează corespunzător codul C, conform unor opțiuni configurabile prin argumente: spațierea și indentarea folosită, introducerea de linii libere, afișarea codului de funcții, pentru blocuri `if` și `for`, afișarea comentariilor.

`indent` permite integrarea diverselor opțiuni în ceea ce se numește stil (`COMMONSTYLES`). În mod implicit se folosește stilul de codare folosit de GNU[1]. Fișierul 14.1 conține cod indentat necorespunzător. Folosind comanda de mai jos se va obține
fișierul 14.2 formatat folosind stilul GNU.

```
1 razvan@valhalla:~/development$ indent ugly-code.c -o ugly-code-default.c
```

Opțiunea `-o` este folosită pentru a specifica fișierul de ieșire. În absența acesteia se modifică direct fișierul sursă.

```
1  #include<stdio.h>
2
3  int main(){ int i;
4   int par=0,impar=0;
5   int   sum= 0;
6   for(i=0;i<100;i++){
7  sum=sum+i;
8  printf("partial␣sum␣is␣%d\n",sum);
9     if(sum%2==0)
10      { printf("par\n");
11        par++;}
12     else {printf("impar\n"); par--;}
13   }
14   return 0;}
```

Listing 14.1: Fișier indentat necorespunzător

```
1  #include<stdio.h>
2
3  int
4  main ()
5  {
6    int i;
7    int par = 0, impar = 0;
8    int sum = 0;
9    for (i = 0; i < 100; i++)
10     {
11       sum = sum + i;
12       printf ("partial␣sum␣is␣%d\n", sum);
13       if (sum % 2 == 0)
14         {
15           printf ("par\n");
16           par++;
```

[1] http://www.gnu.org/prep/standards/

```
17          }
18        else
19          {
20            printf ("impar\n");
21            par--;
22          }
23      }
24    return 0;
25  }
```

Listing 14.2: Fișier formatat cu stilul GNU

Folosind alte opțiuni se pot specifica alte stiluri de coding: `-kr` pentru stilul Kernighan & Ritchie, `-linux` pentru stilul Linux, `-orig` pentru stilul inițial BSD.

Fișierul 15.8 este formatat folosind stilul Linux, folosind comanda:

```
razvan@valhalla:~/development$ indent -linux ugly-code.c -o ugly-code-linux.c
```

```
#include<stdio.h>

int main()
{
        int i;
        int par = 0, impar = 0;
        int sum = 0;
        for (i = 0; i < 100; i++) {
                sum = sum + i;
                printf("partial sum is %d\n", sum);
                if (sum % 2 == 0) {
                        printf("par\n");
                        par++;
                } else {
                        printf("impar\n");
                        par--;
                }
        }
        return 0;
}
```

Listing 14.3: Fișier formatat cu stilul Linux

Utilitarul `astyle`[1] (*Artistic Style* permite indentarea și formatarea codului C, C++, C# și Java. La fel ca și `indent`, `astyle` are diverse opțiuni de formatare și stiluri predefinite. O lipsă a utilitarului astyle este absența opțiunilor de introducere de spații. Comenzile de mai jos formatează un fișier conform stilurilor GNU, Linux sau Kernighan & Ritchie:

```
razvan@valhalla:~/development$ astyle --style=gnu < ugly-code.c > ugly-code-astyle-gnu.c

razvan@valhalla:~/development$ astyle --style=linux < ugly-code.c > ugly-code-astyle-linux.c

razvan@valhalla:~/development$ astyle --style=k\&r < ugly-code.c > ugly-code-astyle-kr.c
```

[1] http://astyle.sourceforge.net/

14.3 Editorul Vim

Editoarele reprezintă utilitarele de bază pentru un dezvoltator de aplicații. Editoarele profesioniste vor oferi acestuia facilități pentru parcurgerea, prelucrarea și editarea rapidă a fișierelor cod sursă. Editoarele profesioniste vor oferi funcționalități de colorare a sintaxei (*syntax highlighting*), colapsarea codului (*code folding*), indentare automată etc.

În lumea Unix, cele mai cunoscute editoare sunt Vim și GNU Emacs. Secțiunea următoare prezintă modul de lucru și funcționalitățile editorului Vim.

Unul dintre cele două editoare protagoniste, tradițional, ale "războiului editoarelor"[1] este **`vi`**. Pentru că programul original nu mai este activ dezvoltat, au apărut numeroase clone, derivate ale codului, sau rescrieri complete. Cel mai cunoscut astfel de program este **`Vim`**[2], dezvoltat de Bram Moolenaar.

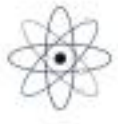
Majoritatea distribuțiilor de Linux vin cu o clonă de **`vi`** minimală preinstalată. Pentru majoritatea exemplelor următoare, aceasta nu este suficientă; recomandăm, deci, instalarea pachetului `vim` întreg. În sistemele Debian, de exemplu, pachetul instalat implicit se numește `vim-tiny`, iar cel cu mai multe opțiuni – `vim`. Pentru varianta care include interfață grafică, există `vim-full`.

Pornirea editorului. Moduri

Vim poate fi pornit din linie de comandă, având drept parametru opțional numele fișierului pe care să îl încarce. Lipsa acestui parametru înseamnă editarea unui fișier nou.

Odată pornit, editorul nu funcționează convențional. El este în modul **Normal**, unde nu este posibilă introducerea textului. Funcție de sarcina care trebuie îndeplinită, Vim poate fi într-unul dintre modurile de mai jos[3]:

- **Normal** – facilitează deplasarea rapidă prin text. De asemenea, o serie de opțiuni avansate ca mark-uri sau fold-uri, descrise sumar în continuare, pot fi manevrate în modul Normal.
- **Insert** – singurul mod în care putem introduce text. Acesta **nu** este modul implicit la pornirea editorului, deci nu este posibilă introducerea de text imediat după încărcarea unui fișier. Această particularitate este sursa multor confuzii. Pentru a intra în modul Insert din Normal, comanda cea mai simplă este `i`. Pentru a reveni în modul normal, se folosește `Escape`.

  Pentru a edita text eficient, nu se rămâne în modul Insert mai mult decât este nevoie.
- **Command** – comenzile avansate se introduc utilizând acest mod. Din modul Normal, se folosește `:` pentru a intra în modul Command. Lansarea comenzii se face folosind `Enter`, iar anularea ei, natural, prin `Escape`. Multe dintre comenzile Vim sunt precedate de `:`. Aceasta înseamnă că introducerea lor se face în modul Command.

[1] http://en.wikipedia.org/wiki/Editor_war
[2] Vi iMproved – http://www.vim.org/
[3] lista nu este exhaustivă

- **Visual** – selectarea unei porțiuni arbitrare de text se face folosind acest mod. După selectare, o comandă va acționa, dacă acest lucru este posibil, asupra textului selectat. Intrarea în modul Visual se face, din Normal, folosind `v`, iar revenirea în Normal – utilizând `Escape` sau lansând o comandă de editare.

Câteva comenzi de bază sunt:

- `:w` – salvare fișier ("write"). Dacă fișierului îi este asociat un nume, se suprascrie vechiul fișier. Dacă comanda este urmată de un nume de fișier, acesta este creat sau, dacă există, suprascris;
- `:q` – ieșire din program ("quit"). Dacă există modificări nesalvate, utilizatorul trebuie fie să le salveze (`:w`), fie să folosească varianta `:q!` pentru a nu le salva;
- `:wq` – scurtătură pentru cele două comenzi de mai sus.

Sistemul integrat de Help

Spre deosebire de majoritatea comenzilor Unix, pagina de manual `vim` nu își propune să documenteze în întregime utilizarea editorului, ci doar detalii despre argumentele în linia de comandă. Vim are un sistem intern de help, foarte dezvoltat.

Pentru a afla informații despre comanda `:w`, se folosește `:help :w`. Implicit, ecranul va fi împărțit în jumătate, cu partea superioară afișând informații de ajutor. Închiderea noii ferestre se face folosind `:q`.

Este util de reținut că se poate folosi `:help` pentru comenzi în mod Normal, comezi "extinse" (caracterul `:` care le precede trebuie inclus), dar și parametri de configurare. Aceștia din urmă trebuie să fie dați între ghilimele, de exemplu: `:help 'cindent'`. Nu în ultimul rând, o resursă excelentă pentru familiarizarea cu Vim este `vimtutor`, un tutorial *self-contained* care vine cu editorul. Lansarea lui se face convențional, în linie de comandă.

Deplasarea în modul Normal

Modul Normal oferă câteva opțiuni inedite de deplasare. Parte din puterea editorului `vi` stă în natura sa modală, pentru că atunci când nu se editează text, tastele alfabetice sunt disponibile pentru alte comenzi utile.

Deși este posibilă, inclusiv în modul Insert, folosirea tastelor săgeți pentru deplasare prin text, modul Normal oferă o alternativă avantajoasă. Se pot folosi tastele `hjkl` în loc de săgeți, după topologia:

```
1    k
2  h j l
```

Aceste taste au avantajul că, pe o tastatură standard QWERTY, se află pe *home row*, rândul central al tastaturii. Ele pot fi apăsate cu mâna dreaptă, fără a deplasa prea mult degetele din poziția lor naturală, folosind tehnica *touch typing*[1].

Alte comenzi utile pentru deplasare, precum și semnificațiile lor, sunt:

[1] http://en.wikipedia.org/wiki/Touch_typing

- `$` – sfârșitul liniei (analog `End`);
- `^` – începutul liniei (analog `Home`);
- `w` – cuvântul următor ("word");
- `b` – cuvântul anterior ("back");
- `fx` – următoarea apariție a caracterului `x` pe linia curentă ("find");
- `Fx` – precedenta apariție a caracterului `x` pe linia curentă;
- `<număr>g` – poziționare pe linia dată ("go to line");
- `gg` – prima linie;
- `G` – ultima linie;
- `%` – paranteza care corespunde celei de sub cursor. Perechile de paranteze pot fi configurate: `:help 'matchpairs'`.
- `/text` – următoarea apariție a textului specificat. Aceasta este o comandă "extinsă" (tastarea `/` provoacă intrarea în modul Command).

Editare avansată

Există comenzi care, lansate în modul Normal sau Visual, manipulează text. Majoritatea celor prezentate mai jos funcționează în felul următor:

- dacă o comandă este lansată în modul Normal, ea așteaptă o secvență de deplasare, pentru a specifica textul asupra căruia să opereze. Astfel, `dfa` va șterge tot textul din poziția curentă până la următoarea apariție a caracterului "a" pe linia curentă;
- dacă o comandă este lansată în modul Visual, ea acționează imediat asupra textului selectat. În plus, se iese din modul Visual și se revine în Normal. Excepție fac comenzile similare cu `c`, descrise mai jos, care duc în modul Insert.

Câteva comenzi de editare și variante utile ale lor sunt prezentate în continuare, cu mențiunea esențială că lista nu este nicidecum completă. Important este că o comandă poate fi urmată de orice mișcare[1], inclusiv folosind repetiții[2].

- `d` – ștergere ("delete"). Textul este pus în "clipboard";
  - `dd` – șterge linia curentă (scurtătură);
  - `D` – șterge restul liniei curente (scurtătură);
  - `dw` – șterge următorul cuvânt (comandă compusă);
  - `dG` – șterge de la linia curentă, până la finalul fișierului (comandă compusă);
- `c` – schimbare ("change"). Textul este pus în "clipboard" și editorul intră în modul Insert;
  - `cc` – schimbă linia curentă (scurtătură);

[1] `:h motion.txt`
[2] `:h repeat.txt`

  - `c` – schimbă restul liniei curente (scurtătură);
  - `c8w` – schimbă următoarele 8 cuvinte (comandă compusă);
  - `c^` – schimbă de la începutul liniei, până la caracterul curent (comandă compusă);
- `y` – copiere în "clipboard" ("yank"). Nu afectează textul, dar poate fi folosit în conjuncție cu `p`;
- `p` – inserare din "clipboard" ("put" sau "paste"). Pune conținutul ultimei comenzi `d`, `c`, sau `y` după poziția curentă.

O greșeală frecventă este aceea de a copia ceva folosind `y`, apoi, înainte de a `p`une conținutul în alt loc, a șterge conținut superfluu, spre exemplu o linie goală. Această ștergere suprascrie conținutul "clipboard"-ului. O soluție imediată este comandă `u` ("undo"), dar răspunsul mai amplu este folosirea registrelor.

Registre

Registrele sunt locuri în memorie unde Vim poate stoca text. Astfel, ceea ce până acum am numit "clipboard" este, de fapt, un registru implicit.

Registrele de uz general sunt adresate folosind literele alfabetului. Nu există nicio diferență între a folosi litere mici sau mari. Modul de folosire a registrelor cu comenzile de editare este următorul:

```
1 "Rc
```

Semnificația componentelor comenzii este următoarea:

- `"` – anunță că vom folosi un registru nestandard;
- `R` – anunță registrul folosit. Poate fi orice registru din domeniul `a-z` sau unul special[1];
- `c` – comanda propriu-zisă. Toate precizările și exemplele secțiunii precedente sunt valabile.

Utilizare `make`

Folosind comanda `:make`, se poate rula **`make`** (vezi secțiunea 11.5) direct din Vim. Implicit, Vim recunoaște mesajele de eroare și le listează, facilitând deplasarea prin surse.

Deplasarea între mesajele de eroare se face utilizând comanzile `:cnext` și `:cprev`. Aceste comenzi funcționează și dacă eroarea nu se află în fișierul curent, deschizând fișiere noi dacă este nevoie. Astfel, Vim poate fi folosit eficient pe post de IDE (vezi secțiunea 14.9).

O altă comandă utilă în acest sens este `:copen`, care deschide o fereastră cu toate erorile. Acesta poate fi folosită pentru poziționarea pe o anumită eroare fără a folosi `:cnext` în mod repetat. Denumirea generică a acestor facilități este `quickfix`.

[1] mai multe informații la `:h registers`

Configurarea Vim

Editorul Vim este remarcat prin flexibilitatea sa, iar elementul central al configurabilității sale sunt opțiunile. Folosind comanda `:set` se pot interoga sau configura opțiuni.

Unele opțiuni sunt de tip boolean. Acestea se setează pe *true* folosind numele lor (`:set autowrite`), iar pe false lipind `no` la începutul numelui (`:set noautowrite`).

O listă completă a lor este prezentă în help, dar poate fi intimidantă. Câteva exemple relevante sunt prezentate în continuare.

- `textwidth` – lungimea unei linii după care se face wrap. În general, pentru text, cod sursă, sau mesaje email, această opțiune trebuie să fie mai mică decât 80. O valoare de 0 înseamnă că Vim nu va face line wrap;
- `cindent` – dacă această opțiune este setată, Vim va face indentare "inteligentă" bazată pe sintaxa limbajului C. Aceasta este o opțiune booloeană;
- `filetype` – tipul fișierului editat. Această opțiune influențează tipul de *syntax highlighting* folosit, dar și alte opțiuni.

Nu în utimul rând, interogarea valorii unei opțiuni se face utilizând comanda `:set`, dar adăugând caracterul `?` după numele opțiunii:

```
1 :set cindent?
```

Configurare persistentă

Configurarea opțiunilor la fiecare rulare a editorului este incomodă. Pentru a păstra unele valori ale opțiunilor, dar și pentru a rula script-uri de extensie, se folosește un fișier de configurare.

Vim citește, în ordine, `/etc/vim/vimrc`, apoi `~/.vimrc`. Astfel, este posibil să suprascriem opțiuni globale folosind fișierul personal de configurare.

Distribuțiile livrează, în general, un fișier de configurare în `/usr/share/vim`. Acesta poate fi copiat în /etc/vim/vimrc și apoi personalizat local. Este important ca unele opțiuni, spre exemplu `syntax`[1], să fie activate tot timpul, pentru folosirea eficientă a editorului.

Există posibilitatea configurării unor opțiuni numai pentru anumite tipuri de fișiere. Linia următoare configurează lungimea liniei, indentarea, și comportamentul înlocuirii spațiilor cu tab-uri numai pentru sursele sau headerele C:

```
1 autocmd FileType c,cpp  set tw=72 cindent noexpandtab
```

Editarea de fișiere multiple

Vim poate edita mai multe fișiere simultan, iar abstracțiile pe care le folosește pentru a prezenta utilizatorilor acest lucru sunt diverse.

[1] `:syntax on`

O posibilitate este împărțirea ecranului în mai multe ferestre, cum face comanda `:help`. Deschiderea unei noi ferestre se face folosind `:split` sau `:vsplit`, urmate de numele fișierului de editat. Deplasarea între ferestre se face utilizând `hjk`, dar precedate de prefixul `Ctrl-W`. Închiderea unei ferestre este un simplu și bine-cunoscut `:q`.

Când mai multe ferestre nu sunt suficiente, vim poate folosi tab-uri. **Fiecare tab poate conține mai multe ferestre**, deci este o greșeală a asocia unu-la-unu fișierele deschise cu tab-urile. Pentru deschiderea unui tab se folosește `:tabnew`, urmat opțional de numele unui fișier, iar pentru deplasare între tab-uri `gt` și `gT`. Închiderea ultimei ferestre dintr-un tab provoacă închiderea tab-ului.

Informații mai complete se găsesc, ca de obicei, în help: `:help windows.txt` și `:help tabpage.txt`.

Lucrul cu proiecte mari

A edita un ansamblu de fișiere mari pune probleme complet diferite față de a adăuga câteva linii unui fișier de configurare. Vim are facilități pentru primul caz, așa cum se va vedea în continuare.

Pentru deplasarea într-un proiect C, Vim are suport de `ctags`[1]. Generarea unui fișier `ctags` se face folosind:

```
1 vlad@cormyr:~/school/so/tema4 $ ctags -aR *
```

Odată generat un astfel de fișier, Vim trebuie notificat de existența lui, prin intermediul opțiunii `tags`. Deplasarea la locul de definiție al funcției de sub cursor se face utilizând `Ctrl-]`, iar pentru revenirea în locul anterior[2] se folosește `Ctrl-T`. O alternativă mai puternică, dar ceva mai dificil de folosit, este `cscope`[3]. Există script-uri Vim care oferă facilități similare pentru cod scris în alte limbaje. În general, ele sunt publicate pe site-ul central Vim[4].

Uneori, este utilă memorarea unui loc într-un fișier. Vim numește acest concept **mark**. Pentru a pune un mark pe linia curentă, se folosește, în modul Normal, `m`, urmat de litera corespunzătoare mark-ului. Literele mici sunt mark-uri per-fișier, iar cele mari sunt globale. Deplasarea până la un mark se face utilizând ' (apostrof) urmat de litera corespunzătoare.

Combinația '' (două apostroafe) este analogul "Alt-Tab". Ea duce cursorul unde a fost înainte, și are euristici care detectează deplasările semnificative. Astfel, deplasările mici sunt ignorate.

O altă facilitate pentru lucrul cu fișiere lungi este **folding**. Aceasta este, în esență, comprimarea *vizuală* (fișierul rămâne neschimbat) a unei regiuni de text. Criteriul după care se face comprimarea poate fi nivelul de indentare, elemente de sintaxă, sau marcaje *ad hoc*. Modul de comprimare este dat de opțiunea `foldmethod`.

Combinații de taste esențiale pentru *folding* sunt:

[1] http://ctags.sourceforge.net/
[2] *popping the tag stack*
[3] http://cscope.sourceforge.net
[4] http://www.vim.org/scripts/index.php

- `zo` – deschide un fold;
- `zc` – închide un fold;
- `zf` – creează un fold, dacă `foldmethod=marker`. Această comandă inserează marcajele configurate cu `foldmarker`.

14.4 Sisteme de control al versiunii

14.4.1 Principii

Sistemele de control al versiunii sunt programe care țin codul sursă și toată istoria modificărilor asupra lui. Folosirea unui astfel de sistem este o practică aproape universală în industria software, dar și o deprindere utilă pentru proiecte personale.

Un sistem de control al versiunii (VCS – *Version Control System*) protejează, în primul rând, împotriva modificărilor care au efecte neprevăzute, dar sunt greu de observat. Dacă, în urma unui build, se constată că s-a introdus un bug, se pot inspecta schimbările făcute și determina mai simplu sursa erorii.

O altă situație care poate apărea este căderea hardware sau ștergerea din greșeală a unor fișiere. Un VCS oferă un backup permanent sincronizat al codului, deci pierderile ar putea fi minimizate.

Nu în ultimul rând, un VCS facilitează colaborarea între dezvoltatori, prin combinarea automată a schimbărilor asupra aceluiași fișier, acolo unde acest lucru este posibil.

Câteva elemente cheie în jargonul VCS sunt:

- **repository** – locul central unde sunt ținute sursele și istoria. În majoritatea cazurilor, repository-ul se află pe un server și este foarte bine protejat. Pierderea unui repository poate fi o catastrofă. Sistemele distribuite (vezi secțiunea 14.4.3) redefinesc noțiunea de repository;
- **working copy** – copia unui repository deținută de un dezvoltator. De obicei, un working copy nu are istoria codului, deci pentru consultarea lor este nevoie de comunicația cu repository-ul. Obținerea unui working copy se face printr-o operație numită tradițional **checkout**;
- **commit** – un set de modificări pe care un dezvoltator îl trimite către repository. Istoria codului este o înșiruire de commit-uri;
- **update** – sincronizarea unui working copy cu repository-ul. Această operație va aplica commit-urile care nu sunt deja în working copy;
- **branch** – o istorie distinctă a sursei. Conceptul este folosit tradițional pentru experimente asupra codului sursă, dar sistemele distribuite redefinesc acest concept.
- **tag** – o stare semnificativă a repository-ului. Un exemplu de tag este un release. O altă situație în care ar fi nevoie de un tag este înaintea unei refactorizări importante.

Se disting două tipuri de sisteme de versionare:

- Sisteme centralizate. Aceastea sisteme se pretează perfect la descrierile conceptelor prezentate mai sus. Sunt încă foarte folosite și activ dezvoltate, dar tendința actuală este migrarea către sistemele distribuite. Exemple relevante sunt CVS, Subversion, Perforce.
- Sisteme distribuite. Ideea din spatele lor este aceea că fiecare working copy are, de fapt, istoria completă a codului. Astfel, nu mai există un repository central. Asemenea sisteme sunt descentralizate, în sensul că nu sunt coerente toate copiile dezvoltatorilor. Din această cauză, o copie a codului se numește branch. Pentru că nu există un repository central, branch-urile sunt adesea schimbate prin email. Exemple relevante de VCS distribuite sunt: git, darcs, Mercurial, Bazaar.

În VCS se țin toate fișierele care nu pot fi generate altfel. Cu alte cuvinte, sistemele VCS mențin surse, dar nu și fișiere binare. O altă regulă este ca, în orice moment, starea repository-ului să fie consistentă; codul trebuie tot timpul să compileze, indiferent dacă are bug-uri sau facilități incomplete. Sintagma "a strica build-ul" descrie un commit care lasă codul într-o stare în care nu se compilează.

14.4.2 Subversion

Subversion[1] este sistemul centralizat cel mai folosit în momentul de față. Este urmașul CVS și adoptă o abordare clasică, simplu de înțeles. Executabilul de Subversion se numește **svn**.

Operația prin care se crează un working copy este `svn checkout`. Subversion poate funcționa peste SSH, sau poate folosi protocolul propriu.

```
1 vlad@cormyr $ svn checkout svn://svn.rosedu.org/wouso
```

Pentru a face un nou commit, se folosește `svn commit`. Schimbările sunt automat transmise serverului central, deci computerul are nevoie de conexiune la rețea. Înainte de a face un commit, se recomandă actualizarea working copy-ului pentru a evita conflictele. Aceast lucru se face folosind comanda `svn update`.

Adăugarea unui nou fișier la repository se face folosind `svn add`. Transferul către server nu se va face, însă, decât la următorul commit.

Folosind `svn status`, se poate inspecta starea copiei de lucru. Această comandă tipărește un sumar al fișierelor schimbate sau adăugate, dar și al celor care nu sunt în repository-ul central ("untracked"). Pentru a vedea schimbările aduse fișierelor, se folosește `svn diff`.

Subversion nu are suport nativ pentru branch-uri și tag-uri, fapt pentru care majoritatea repository-urilor conțin, în rădăcină, 3 directoare:

- `branches` – simulează funcționarea branch-urilor din alte VCS-uri. Practic, crearea unui branch este o copiere a surselor într-un nou subdirector al `branches`;
- `tags` – simulează existența tag-urilor. Crearea unui tag este echivalentă cu o copiere, cu mențiunea că există o convenție ca nimeni să nu modifice conținutul directorului `tags`

[1] http://subversion.tigris.org/

- `trunk` – aici rezidă codul propriu-zis. Acest director este sursa copierilor care fac tag-uri și branch-uri.

14.4.3 Git

Git[1] este un VCS distribuit dezvoltat, inițial, de Linus Torvalds. Se distinge prin flexibilitate și viteză și are o comunitate activă. Este sistemul de versionare folosit de kernelul Linux.

Pentru a folosi eficient Git, se recomandă activarea culorilor și configurarea globală a numelui și adresei de mail a utilizatorului.

```
1 vlad@cormyr $ git config --global user.name "John Doe"
2
3 vlad@cormyr $ git config --global user.email "john.doe@foobar.com"
4
5 vlad@cormyr $ git config --global color.ui always
```

Operația de creare a unui "working copy" (numit branch în terminologie Git) este `git clone`. Pentru comunicația cu serverul se pot folosi HTTP sau SSH, dar este recomandat protocolul propriu Git, care are suport mai bun de compresie.

```
1 vlad@cormyr $ git clone git://git.rosedu.org/rtt.git
```

După modificarea fișierelor, ele trebuiesc adăugate explicit și apoi creat un commit. În cultura Git, acest lucru se sumarizează prin "Git tracks changes, not files". `git commit` lansează un editor în care este solicitată o descriere a noului commit.

```
1 vlad@cormyr $ git add binops.c
2
3 vlad@cormyr $ git commit
```

Starea branch-ului curent poate fi inspectată folosind `git status`, iar schimbările pot fi propagate către branch-ul de unde s-a făcut clonarea ("origin") folosind `git push`. Sincronizarea cu origin se face folosind `git pull`.

Aspecte netriviale ale utilizării Git sunt lucrul cu branch-uri, operațiile de merge, rezolvarea conflictelor, sau folosirea `git rebase`.

14.5 Analiza și parcurgerea codului

Un dezvoltator software profesionist va petrece, de regulă, mult mai mult timp în parcurgerea, înțelegerea și revizia fișierelor cod sursă decât în crearea de cod. Altfel spus, operația de citire a codului este mai frecventă și mai de durată decât cea de scriere.

Pentru a facilita înțelegerea codului, cel care scrie codul va trebui să urmărească un stil de codare sau un set de norme. În același timp, însă, cel care urmărește sau revizuiește codul dispune de un set de utilitare specifice acestui scop. În momentul în care proiectul software dispune de multe fișiere sursă, este foarte important ca revizorul să ajungă rapid la anumite poziții din cadrul fișierelor sursă.

[1] http://git-scm.com/

Utilitare precum `ctags` ajută revizorul să ajungă în locul unde a fost definită o funcție sau o variabilă din doar câteva combinații de taste. Revizorul poate urmări astfel setul de apeluri din cadrul unei aplicații, sau locurile în care o anumită variabilă este folosită sau utilizată. Unele IDE-uri (precum Microsoft Visual Studio (vezi secțiunea 14.12.1).

14.5.1 ctags

Utilitarul `ctags` este folosit pentru a genera un index al simbolurilor dintr-un set de fișiere sursă pentru parcurgerea ușoară a acestora. Indexul este generat într-un fișier **tag** care va fi ulterior folosit de un editor.

Utilitarul are asociate două comenzi: **`ctags`** și **`etags`**. Comanda **`ctags`** este folosită pentru a genera fișierul index pentru a fi folosit de editoare din familia **vi**, iar comanda **`etags`** generează fișier index pentru editoare din familia **Emacs**. Editoarele folosesc comenzi specializate pentru a găsi rapid locul în care este definită/utilizată o funcție sau variabilă pe baza fișierului index.

În general, pentru a genera un fișier index se folosește opțiunea `-R` pentru a parcurge recursiv toate fișierele sursă din directorul curent. Astfel, pentru a genera fișierele index pentru sursele bibliotecii standard C, se vor folosi comenzile:

```
1 razvan@valhalla:~/packages/glibc-2.7/glibc-2.7$ ctags -R .
2
3 razvan@valhalla:~/packages/glibc-2.7/glibc-2.7$ etags -R .
4
5 razvan@valhalla:~/packages/glibc-2.7/glibc-2.7$ ls -l tags TAGS
6 -rw-r--r-- 1 razvan razvan 5338062 Sep 26 11:31 TAGS
7 -rw-r--r-- 1 razvan razvan 7364195 Sep 26 11:31 tags
```

În mod implicit, **`ctags`** generează fișierul `tags`, iar **`etags`** generează fișierul `TAGS`. Fișierul este încărcat automat de editor în cazul în care fișierul index se găsește în directorul curent. Altfel, fișierul poate fi încărcat în Vim folosind o comandă de forma

```
1 :set tags+=/usr/include/tags
```

sau în Emacs, folosind

```
1 M-x visit-tags-table
```

Detalii despre utilizarea fișierelor index în Vim pentru parcurgerea rapidă a codului se găsesc în secțiunea 14.5.1. Pentru Emacs, informații utile sunt descrise în secțiunea asociată a manualului[1].

14.5.2 Analiza statică a codului – splint

Pe lângă erorile raportate de compilator, sau erorile/problemele din momentul execuției raportate de utilitare precum Valgrind (vezi secțiunea 14.8.2), dezvoltatorul poate folosi utilitare de analiză statică a codului (*Static code analysis*[2]). Aceste utilitare parcurg codul sursă fără compilarea, interpretarea sau executarea acestuia.

[1] http://www.gnu.org/software/emacs/manual/html_node/emacs/Tags.html#Tags
[2] http://en.wikipedia.org/wiki/Static_code_analysis

Un astfel de utilitar este **Splint**[1]. Descendent al utilitarului **lint** din Unix, Splint permite detectarea potențialelor probleme de securitate și a erorilor de codare. Splint poate fi configurat să folosească diverse niveluri de verificare. Conform paginii de manual, se oferă un premiu special persoanei care creează un program util care nu raportează erori în cazul folosirii opțiunii `-strict` (verificare strictă).

Un exemplu de rulare a comenzii **`splint`** și o parte a ieșirii acesteia este prezentat mai jos:

```
1  razvan@valhalla:/tmp$ splint -warnposix mini-shell.c
2  Splint 3.1.2 --- 20 Feb 2009
3
4  mini-shell.c: (in function simple_cmd)
5  mini-shell.c:38:5: Return value (type int) ignored: close(pin[0])
6    Result returned by function call is not used. If this is intended, can
   cast
7    result to (void) to eliminate message. (Use -retvalint to inhibit
   warning)
8  mini-shell.c:52:7: Test expression for if not boolean, type int:
9                        cmd->io_flags \& 0x01
10   Test expression type is not boolean or int. (Use -predboolint to
   inhibit
11   warning)
12 mini-shell.c:65:5: Assignment of int to char: str[0] = 0
13   Types are incompatible. (Use -type to inhibit warning)
14 mini-shell.c:68:48: Possibly null storage passed as non-null param:
15                       strlen (getenv(par->string))
16 mini-shell.c:25:5: Function exported but not used outside mini-shell:
17                     simple_cmd
18   A declaration is exported, but not used outside this module.
   Declaration can
19   use static qualifier. (Use -exportlocal to inhibit warning)
20    mini-shell.c:171:1: Definition of simple_cmd
21 [...]
```

După cum reiese din exemplul de mai sus, câteva dintre probleme de codare, neraportate implicit de compilator, sunt:

- ignorarea valorii de retur a unei funcții;
- folosirea unei expresii care întoarce o valoare întreagă în locul unei valori booleene;
- inițializarea unei variabile cu o valoare de alt tip;
- posibila folosire a unei valori null;
- definirea unei funcții non-statice care nu este folosită într-un alt modul.

O bună parte din problemele raportate de Splint nu sunt relevante în contextul unui program. De aceea, Splint pune la dispoziția dezvoltatorului opțiuni de dezactivare a anumitor mesaje, precum `-exportlocal`, `-predboolinit`, `-retvalint` etc.

În general, informațiile oferite de Splint pot fi foarte stricte și fără importanță deosebită în cadrul programului. De aceea, se recomandă selectarea informațiilor furnizate după relevanța acestora.

[1] http://www.splint.org/

14.6 Automatizarea compilării

Este posibil ca unele aplicații să nu se găsească în repository-urile distribuției folosite. Mai mult, este posibil ca acestea să nu fie disponibile într-un format ce ar facilita instalarea (cum ar fi .deb, .rpm). Astfel de aplicații trebuie compilate explicit pentru platforma folosită, pornind de la sursele lor.

14.6.1 Compilarea unei aplicații din surse

De cele mai multe ori, sursele unei aplicații se găsesc în formatul tar.gz sau tar.bz2. Așadar, primul pas este extragerea surselor din aceste arhive folosind utilitarul tar. Presupunând că arhiva cu sursele se numește foo.tar.gz sau foo.tar.bz2, pentru a extrage sursele trebuie rulată una din comenzile:

```
1 user@sys:~$ tar -xzvf foo.tar.gz
2 [...]
3 user@sys:~$ tar -xjvf foo.tar.bz2
```

Următorul pas îl reprezintă obținerea fișierului makefile. În acest punct este recomandată citirea fișierelor README sau INSTALL pentru a vedea exact pașii ce trebuie urmați. Deși pot apărea mici diferențe, majoritatea aplicațiilor pot fi instalate urmărind pașii descriși în continuare.

Scriptul configure este cel responsabil de generarea makefile-ului, acesta verificând versiunile de software disponibile și dacă diverse dependințe sunt îndeplinite. Este important de menționat că prin modificarea acestui script sau prin parametrii pasați în linia de comandă, se poate specifica directorul în care urmează să fie instalată aplicația. Scriptul se găsește în directorul în care am extras sursele, și rularea lui se face astfel:

```
1 user@sys:~/foo$ ./configure
```

Orice fel de eroare întâlnită la acest pas, cum ar fi lipsa unei alte aplicații, va fi semnalată în output-ul comenzii și trebuie rezolvată înainte de trecerea la următoarea etapă.

În urma scriptului de configurare a rezultat un fișier numit makefile (sau Makefile, depinde de fișierul de configurare), pe care îl vom folosi pentru a compila sursele. Pentru a face acest lucru vom folosi utilitarul make, care caută în directorul curent un fișier makefile pe care îl va folosi în obținerea executabilelor:

```
1 user@sys:~/foo$ make
```

Astfel, după rularea make, sursele vor fi compilate însă executabilele rezultate se află doar în directorul curent. Pentru a le muta în directorul de instalare:

```
1 user@sys:~/foo$ make install
```

În acest moment, aplicația a fost instalată și dacă directorul în care aceasta se află există în PATH, ar trebui să se poată rula din directorul curent.

Pentru a scăpa de fișierele temporare ce au fost generate în procesul de configurare și compilare, se poate rula:

```
1 user@sys:~/foo$ make clean
```

Pentru dezinstalarea aplicației, pot fi șterse manual fișierele din directorul unde a fost instalată, sau se poate folosi makefile-ul generat anterior, rulându-se:

```
1 user@sys:~/foo$ make uninstall
```

GNU Autotools

GNU Autotools[1] reprezintă un set de utilitare din cadrul proiectului GNU ce ajută la creșterea portabilității pachetelor software și a simplificării procesului de instalare a acestora. Cum am văzut anterior, majoritatea aplicațiilor pot fi instalate urmând cei trei pași simpli : ./configure, make, make install. Acest lucru se datorează în mare parte utilitarelor din Autotools, care reușesc să uniformizeze procesul de build, indiferent de platformă.

Utilitarele care fac parte din Autotools sunt:

- Automake – pornind de la un fișier numit Makefile.am, creează un fișier Makefile.in, folosit apoi de scriptul de configurare pentru a genera Makefile-ul final;
- Autoconf – este cel care creează scriptul de configurare, pornind de la un fișier numit configure.ac;
- Libtool – abstractizează procesul de creeare a librăriilor statice și dinamice.

14.7 Execuția unui program

În urma procesului de compilare și link-editare rezultă un executabil (vezi secțiunea 11.1.2). Acest executabil deține informații necesare sistemului de operare la rulare, pentru crearea unui proces.

14.7.1 Zonele de memorie ale unui executabil și proces

Un fișier executabil are un format specific. Pe lângă datele de organizare internă, un executabil conține informații despre instrucțiunile care vor fi executate pe procesor și datele folosite. Aceste informații se vor stoca în memorie în momentul creării și rulării procesului asociat. Un executabil va conține, așadar, **zona de cod** și **zonele de date**.

Zona de cod, denumită și zona text este, de fapt, traducerea în cod mașină a funcțiilor scrise în cod sursă. Aceste instrucțiuni vor fi executate pe rând pe procesor, în momentul creării procesului. Funcția main dintr-un program C este punctul de start din zona de cod.

Zonele de date reprezintă echivalentul binar al variabilelor globale definite de un programator în codul sursă. Există mai multe zone de date:

- `zona pentru date inițializate` (*.data*) unde sunt stocate variabilele globale inițializate (de forma `int size = 30;`)

[1] http://sourceware.org/autobook/autobook/autobook.html

- `zona pentru date neinițializate` (`.bss`) unde sunt stocate variabilele globale neinițializate (de forma `int file_array[10];`)
- `zona pentru date read-only` (`.rodata`) unde sunt stocate datele ce pot fi doar citite; din această categorie fac parte literalii din C; de exemplu, în cazul instrucțiunii `printf("Hello, World!\n");`, șirul `Hello, World\n` este stocat în `.rodata`.

În momentul creării procesului, se alocă memorie pentru zonele de mai sus. În afara acestor zone, în cadrul unui proces se alocă zone de memorie doar pe parcursul rulării (zone de memorie dinamice), care nu sunt disponibile în executabil. Cele mai importante zone sunt stiva și heap-ul.

Stiva (*stack*) este zona de memorie dinamică folosită pentru a reține informații despre apelurile de funcții și pentru a stoca variabilele locale funcțiilor. La fiecare apel de funcție, pe stivă se creează un cadru de stivă (*stack frame*) cu informații despre apel și pentru alocarea variabilelor locale funcției. În momentul în care se revine din funcții, cadrul de stivă asociat este eliberat de pe stivă. Motivul pentru care se recomandă evitarea apelurilor recursive este încărcarea stivei în cazul unui număr mare de apeluri.

Heap-ul este zona de memorie folosită pentru alocări dinamice. Alocările dinamice sunt alocările care se petrec în timpul rulării procesului (la *runtime*). Opusul este dat de alocările statice, care sunt realizate în momentul compilării. Variabilele globale sunt alocate static în zona `.data` sau în zona `.bss`.

Alocarea dinamică se realizează, în C, cu ajutorul apelurilor de bibliotecă `malloc`, `calloc` sau `realloc`. Acestea primesc ca argument numărul de octeți pentru regiunea care va fi alocată. O greșeală frecventă este omiterea eliberării zonelor de memorie alocate. Orice zonă de memorie alocată folosind apelurile din familia malloc va trebui dezalocată folosind apelul `free` atunci când nu mai este folosită. Dacă se omite operația de eliberare a memoriei se poate ajunge la situații de *memory leaking*[1] care afectează negativ funcționarea aplicației sau a sistemului.

14.7.2 Utilizarea bibliotecilor partajate

În momentul execuției, sistemul de operare efectuează o serie de operații pentru a pregăti aplicația pentru rulare. Procesele lucrează în zone de memorie independente unele de altele. Astfel, ceea ce se găsește la o anumită adresă pentru un proces nu se găsește și la alt proces.

Pe lângă încărcarea zonei de cod și a zonei de date din executabil, sistemul de operare trebuie să încarce și bibliotecile partajate. Bibliotecile statice devin parte din aplicații și nu au cum să fie tratate la execuție.

Pentru a vizualiza lista bibliotecilor partajate utilizate de un program, se folosește comanda **`ldd`**, precum în exemplul următor:

```
1 mircea@cougar:~/carte-uso/cap-10$ ldd /bin/bash
2         linux-gate.so.1 =>  (0xffffe000)
3         libreadline.so.5 => /lib/libreadline.so.5 (0xb7eb4000)
4         libhistory.so.5 => /lib/libhistory.so.5 (0xb7eac000)
```

[1] http://en.wikipedia.org/wiki/Memory_leak

```
5         libncurses.so.5 => /lib/libncurses.so.5 (0xb7e6a000)
6         libdl.so.2 => /lib/libdl.so.2 (0xb7e66000)
7         libc.so.6 => /lib/libc.so.6 (0xb7d3a000)
8         /lib/ld-linux.so.2 (0xb7f0d000)
```

Informațiile despre bibliotecile partajate aflate în sistem sunt colectate (manual sau automat) utilizând programul **`ldconfig`**. Această comandă:

- caută biblioteci partajate în locațiile standard și în directoarele specificate în /etc/ld.so.conf,
- configurează legături simbolice corecte în directoarele cu biblioteci (pentru a oferi denumiri standard biblitecilor, din punct de vedere al formatului)
- salvează referințele către biblioteci într-un cache.

Rularea programului **`ldconfig`** la fiecare bootare este ineficientă – acesta este un motiv în plus pentru utilizarea unui cache.

Modul în care programele utilizează bibliotecile de funcții este definit la link-editare. Linker-ul (**`ld`**) caută bibliotecile partajate în directoarele standard și în directoarele adăugate în linia de comandă prin intermediul unor parametri speciali (`-rpath dir` și `-Ldir`).

Într-un sistem bazat pe biblitecile standard GNU C, inclusiv toate sistemele Linux, pornirea unui executabil de tip ELF (formatul standard al executabilelor) determină încarcarea și rularea unui program loader. Pe sistemele Linux, acesta este `/lib/ld-linux.so.X` (unde X este versiunea). Acest loader găseste și încarcă bibliotecile partajate utilizate de către program.

Dacă se dorește supraîncărcarea unor funcții dintr-o bibliotecă dar cu păstrarea restului bibliotecii, se pot introduce bibliotecile supraîncarcate în `/etc/ld.so.preload` – aceste biblioteci "preîncărcate" vor avea întâietate relativ la setul standard. Acest fișier pentru preîncărcări este utilizat de obicei pentru patch-uri de urgență – distribuțiile nu includ de obicei un astfel de fișier.

14.7.3 Analiza apelurilor de sistem și a semnalelor

Pentru a interacționa cu sistemul de operare (cel care partajează resursele computerului), bibliotecile de funcții apelează o serie de funcții ale kernel-ului numite apeluri de sistem (system calls).

În continuare sunt prezentate doar câteva dintre utilizările apelurilor de sistem:

- deschiderea/inchiderea unui fișier (**`open`**, **`close`**)
- citirea/scrierea dintr-un/intr-un fișier (**`read`**, **`write`**)
- deschiderea unui socket (pentru o conexiune TCP/IP)
- executia unui program (**`execv`**)

Aceste funcții sunt implementate la nivelul nucleului sistemului de operare deoarece SO-ul face management-ul resurselor. Bibliotecile de functii "îmbracă" aceste apeluri în funcții/clase/obiecte utilizabile direct de către programatori.

strace

Comanda strace este utilă pentru urmărirea apelurilor de sistem efectuate la execuția unui program. Această analiză reprezintă o metodă destul de eficientă în debugging-ul programelor al căror cod sursa nu-l avem.

Pentru a observa ce informații poate să ofere **strace**, se va analiza ieșirea programului pentru o aplicație simplă (o parte din ieșirea programului a fost eliminată pentru a ușura analiza):

```
1  mircea@cougar:~/carte-uso/cap-10$ strace echo "USO"
2  execve("/bin/echo", ["echo", "USO"], [/* 51 vars */]) = 0
3  uname({sys="Linux", node="aquarium", ...}) = 0
4  [...]
5  access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or
   directory)
6  open("/etc/ld.so.cache", O_RDONLY)      = 3
7  fstat64(3, {st_mode=S_IFREG|0644, st_size=141679, ...}) = 0
8  [...]
9  close(3)                                = 0
10 open("/lib/tls/libc.so.6", O_RDONLY)    = 3
11 read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\220R\1"..., 512)
    = 512
12 fstat64(3, {st_mode=S_IFREG|0755, st_size=1363203, ...}) = 0
13 [...]
14 close(3)                                = 0
15 [...]
16 open("/usr/lib/locale/locale-archive", O_RDONLY|O_LARGEFILE) = 3
17 fstat64(3, {st_mode=S_IFREG|0644, st_size=38399616, ...}) = 0
18 [...]
19 close(3)                                = 0
20 [...]
21 fstat64(1, {st_mode=S_IFCHR|0600, st_rdev=makedev(136, 1), ...}) = 0
22 [...]
23 write(1, "USO\n", 4USO
24 )                       = 4
25 [...]
26 exit_group(0)                           = ?
```

Se poate observa:

- primul apel de sistem (**execve**) indică ce parametri a primit programul: echo și USO
- programul a încercat să deschidă fișierul `/etc/ld.so.preload` dar nu a fost găsit (scopul acestui fișier este descris în subcapitolul dedicat bibliotecilor)
- apelul de sistem **open** a returnat valoarea 3 – aceasta reprezintă un descriptor de fișier (fd – file descriptor) – descriptorul de fișier este folosit în procesele de citire și în procesele de scriere (e un fel de pointer la fișier, pointer utilizat de sistemul de operare)
- apelul de sistem **close** are ca singur parametru – descriptorul de fișier care trebuie închis – odată închis, un descriptor de fișier poate fi refolosit
- apelul de sistem **read** indică o citire din descriptorul de fișier 3
- apelul de sistem **write** are 3 parametri:

- 1 = file descriptor-ul unde se scrie: în cazul nostru, după cum știm, 1 este file descriptor-ul pentru stdout (ieșirea standard)
- USO\n = acesta este textul ce trebuie afișat de comanda echo
- 4 = lungimea textului ce trebuie afisat

În acest program s-au putut identifica câteva apeluri de sistem frecvente.

14.8 Depanarea unui program

De cele mai multe ori, programele scrise de noi nu funcționează corect de la prima rulare. De asemenea, se poate întâmpla ca un program care a funcționat corect timp de câțiva ani înainte să nu mai ruleze bine acum, în momentul introducerii unei date neprevăzute initial.

Depanarea unui program este o metodă de a găsi și identifica aceste erori sau defecte din program. Depanarea este o activitatea de durată și obositoare, de obicei. Abilitatea programatorului de a identifica *bug-urile* (greșelile din program) reprezintă factorul cheie în acest proces. Dar, nu trebuie uitată influența pe care o au metodele de depanare (*debugging*).

Desigur, pentru programele scurte se poate folosi *printf* pentru afișarea diverselor informații chiar dacă este o metodă ce presupune modificarea codului mai mult decât este nevoie pentru corectarea greșelii.

Pentru programele mai complicate este necesară folosirea unor instrumente de depanare, așa numitele *debuggers*. Acestea sunt programe sau biblioteci ce-i permit programatorului monitorizarea execuției unui program, oferindu-i capacitatea de a-l porni, opri, reporni și, în unele cazuri, a rula înapoi.

14.8.1 gdb, ddd

Un bun instrument de debug este GNU Debugger cunoscut mai mult sub denumirea prescurtată de GDB[1], depanatorul standard GNU, scris de Richard Stallman în 1986. Este un instrument portabil, capabil de a depana programe scrise într-o suită mai largă de limbaje (C, Ada, Basic, Fortran, C++ etc).

GDB se remarcă prin următoarele 4 lucruri, capacități cerute de la orice program de depanare a codului:

- se permite pornirea execuției unui program, specificând orice parametru ce-i poate afecta comportamentul;
- este posibilă oprirea programului într-un anumit punct, pe baza unor condiții specificate de utilizator;
- în cazul opririi premature a programului, se pot examina toate condițiile ce au dus la acest caz;

[1] http://sourceware.org/gdb

- se permite schimbarea valorii unei variabile – uneori putându-se schimba și instrucțiuni – pentru a modifica execuția programului fără a părăsi depanatorul.

Pentru a putea depana un program cu GDB, acesta trebuie întâi compilat utilizând un flag special *-g* pentru a se permite salvarea anumitor informații ce vor fi utile în continuare. De asemenea, se recomandă evitarea oricărei optimizări.

În continuare, vom prezenta un exemplu de rulare a unui program sub gdb. Pornim prin a lista problema pe care o avem:

```
mihai@keldon:~/tmp$ cat 1.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
        int i, n, **a, j;
        n = 6;
        a = (int*) malloc(n * sizeof(char));
        for (i = 0; i < n; i++){
                for (j = 0; j < n; j++)
                        a[i][j] = i + (j-1);
        }

        for (i = 0; i < n; i++){
                for (j = 0; j < n; j++)
                        printf("%d ", a[i][j]);
        printf("\n");
        }
        return 0;
}

mihai@keldon:~/tmp$ gcc -Wall 1.c
1.c: In function 'main':
1.c:8: warning: assignment from incompatible pointer type

mihai@keldon:~/tmp$ ./a.out
Segmentation fault
```

Deși este normal să tratăm întâi warningurile și apoi să depanăm codul, vom proceda diferit acum, ignorând outputul lui gcc.

```
mihai@keldon:~/tmp$ gcc -Wall -g -O0 1.c
...
mihai@keldon:~/tmp$ gdb ./a.out
...
(gdb) run
Starting program: /home/mihai/tmp/a.out

Program received signal SIGSEGV, Segmentation fault.
0x08048488 in main () at 1.c:11
11                              a[i][j] = i + (j-1);
(gdb) list
6               int i, n, **a, j;
7               n = 6;
8               a = (int*) malloc(n * sizeof(char));
(gdb) print *a
$4 = (int *) 0x0
(gdb) break main
```

```
18 Breakpoint 1 at 0x8048445: file 1.c, line 7.
19 (gdb) r
20 The program being debugged has been started already.
21 Start it from the beginning? (y or n) y
22 Starting program: /home/mihai/tmp/a.out
23
24 Breakpoint 1, main () at 1.c:7
25 7               n = 6;
26 (gdb) next
27 8               a = (int*) malloc(n * sizeof(char));
28 (gdb) display a
29 1: a = (int **) 0xbfe62388
30 (gdb) n
31 9               for (i = 0; i < n; i++){
32 1: a = (int **) 0x8634008
33 (gdb)
34 10                      for (j = 0; j < n; j++)
35 1: a = (int **) 0x8634008
36 ...
37 (gdb) q
38 The program is running.  Exit anyway? (y or n) y
```

În interiorul gdb, s-au folosit următoarele comenzi:

- run, prescurtat r – permite execuția programului;
- list, prescurtat l – permite listarea codului programului în jurul punctului unde acesta s-a oprit;
- print, prescurtat p – permite afișarea unei variabile;
- break, prescurtat b – permite stabilirea unui punct de întrerupere în interiorul programului;
- next, prescuratat n – execută următoarea instrucțiune a programului (cea afișată în momentul tastării comenzii);
- display, disp – permite afișarea unei variabile pe parcursul procesului de depanare;
- quit, q – terminarea depanării programului, ieșirea din gdb.

Observați că simpla trimitere a unei comenzi goale (doar apăsarea tastei Enter) duce la executarea comenzii anterioare.

Pentru mai mult detalii, consultați manualul GDB[1],.

ddd

Deși foarte util, GDB are un dezavantaj major: neavând interfață grafică, folosirea lui este mai greoaie. Din fericire, există un front-end grafic sub denumirea de Data Display Debugger (DDD[2]).

Pe lângă oferirea unei interfețe grafice ce permite stabilirea mult mai ușoară a unor puncte de oprire a programului (*breakpoints*), ddd s-a remarcat prin afișarea interactivă

[1] http://sourceware.org/gdb/current/onlinedocs/gdb_toc.html#SEC_Contents
[2] http://www.gnu.org/software/ddd/

a datelor programului utilizând grafuri. Această afișare permite studierea unui program prin analiza datelor și nu a liniilor de cod, un proces evident mult mai simplu.

```
mihai@keldon:~/tmp$ gcc -Wall -g -O0 1.c
...
mihai@keldon:~/tmp$ ddd ./a.out
...
```

Fiind un front-end al GDB, principiile de folosire sunt similare. Se va compila sursa și se va porni debugger-ul, restul comenzilor fiind înlocuite cu butoane în mediul grafic.

14.8.2 valgrind

Valgrind[1] este un utilitar de programare folosit pentru a depana memoria, a detecta leak-urile de memorie și a face profiling. Autorul lui este Julian Seward, care în 2006 a câștigat un al doilea Google-O'Reilly Open Source Award pentru Valgrind.

Valgrind este un framework de analiză dinamică. El conține un set de utilitare, iar fiecare dintre acestea realizează un tip aparte de depanare sau profiling. Dintre aceste componente, cele mai importante sunt:

- memcheck – checker de memorie.
- cachegrind – simulator de cache, numărul de instrucțiuni executate și cache miss-uri cauzate
- helgrind – depistează posibile race conditions din program
- massif – un profiler pentru heap, spune cât memory heap utilizează programul

Cel mai des folosit este memcheck, de aceea implicit valgrind îl folosește pe acesta, dacă se dorește folosirea altui utilitar acesta se va da ca parametru.

```
andrew@Goliath:~$ valgrind --tool=massif ./test

andrew@Goliath:~$ valgrind --tool=cachegrind ./test
```

Similar cu gdb, programul trebuie compilat cu flag-ul *-g* pentru a primit informații de depanare, astfel încât să se poată specifica linia exactă la care a apărut o problemă. Compilarea fără optimizări este și ea recomandă dacă este posibil, dacă acest lucru ar determina un timp de rulare prea mare se recomandă *-O1*. Folosirea unei optimizări de la *-O2* în sus poate produce erori de valori neinițializate care nu sunt corecte.

În continuare vom analiza un exemplu de rulare. Dacă în mod normal ați rula programul astfel:

```
andrew@Goliath:~$ ./test arg1 arg2
```

Pentru a rula în valgrind cu verificarea leak-urilor de memorie rulați următoarea comandă:

```
andrew@Goliath:~$ valgrind --leak-check=full ./test arg1 arg2
```

Utilitarul memcheck depistează următoarele tipuri de erori:

[1] http://en.wikipedia.org/wiki/Valgrind

- variabile neinițializate
- citiri/scrieri în zone de memorie eliberate
- citiri/scrieri în zone inaccesibile
- leak-uri de memorie

Exemple de output pentru anumite erori:

- citire invalidă

```
Invalid write of size 1
    at 0x804841E: main (example2.c:6)
Address 0x1BA3607A is 0 bytes after a block of size 10 alloc'd
    at 0x1B900DD0: malloc (vg_replace_malloc.c:131)
    by 0x804840F: main (example2.c:5)
```

- utilizare de variabile neinițializate

```
Conditional jump or move depends on uninitialised value(s)
   at 0x402DFA94: _IO_vfprintf (_itoa.h:49)
   by 0x402E8476: _IO_printf (printf.c:36)
   by 0x8048472: main (tests/manuel1.c:8)
```

- free-uri ilegale

```
Invalid free()
   at 0x4004FFDF: free (vg_clientmalloc.c:577)
   by 0x80484C7: main (tests/doublefree.c:10)
```

- suprapunerea adresei destinație cu cea sursă la copierea unor blocuri de memorie

```
Source and destination overlap in memcpy(0xbffff294, 0xbffff280, 21)
    at 0x40026CDC: memcpy (mc_replace_strmem.c:71)
    by 0x804865A: main (overlap.c:40)
```

- rezumat al leak-urilor de memorie

```
LEAK SUMMARY:
   definitely lost: 48 bytes in 3 blocks.
   indirectly lost: 32 bytes in 2 blocks.
     possibly lost: 96 bytes in 6 blocks.
   still reachable: 64 bytes in 4 blocks.
        suppressed: 0 bytes in 0 blocks.
```

Pentru mai multe detalii se va folosi parametrul *--leak-check=full*. Informațiile detaliate despre leak-uri vor arăta astfel:

```
8 bytes in 1 blocks are definitely lost in loss record 1 of 14
   at 0x........: malloc (vg_replace_malloc.c:...)
   by 0x........: mk (leak-tree.c:11)
   by 0x........: main (leak-tree.c:39)

88 (8 direct, 80 indirect) bytes in 1 blocks are definitely lost in
loss record 13 of 14
   at 0x........: malloc (vg_replace_malloc.c:...)
   by 0x........: mk (leak-tree.c:11)
   by 0x........: main (leak-tree.c:25)
```

Site-ul proiectului conține un manual de utilizare[1] în care se pot găsi mai multe informații despre erori[2] și opțiuni[3] pentru memcheck.

Valgrind rulează pe majoritatea platformelor Linux, există și portări neoficiale pentru NetBSD și FreeBSD.

14.9 Medii integrate de dezvoltare

Un mediu integrat de dezvoltare este construit pentru a spori productivitatea programatorilor prin oferirea unor componente strâns legate prin intermediul interfeței cu utilizatorul. Totuși, deoarece este vorba de un sistem complex, productivitatea sporită poate apărea după o perioadă lungă de adaptare și învățare a mediului.

De regulă, un mediu de dezvoltare (IDE) reprezintă un singur program în care se desfășoară tot procesul de dezvoltare: editare, compilare, depanare etc.

În mod obișnuit, un IDE este orientat pe un limbaj de programare specific, deși există și excepții.

14.9.1 Eclipse

Utilizând Java, având un comportament configurabil prin plugin-uri, mediul de dezvoltare Eclipse[4] poate fi utilizat pentru a dezvolta cod într-o mulțime de limbaje.

Sistemul de plugin-uri nu este doar un mecanism ce permite folosirea mai multor limbaje de programare, fiind posibilă și integrarea soluțiilor de versionare a codului (SVN), scrierea documentației LaTeX, testarea aplicațiilor de rețea etc.

Cu toate acestea, Eclipse are și câteva dezavantaje. Interfața grafică este puțin mai complicată. Fiind scris în Java, este absolut necesară prezența unei mașini virtuale Java în sistem pentru a putea fi folosit. Nu în ultimul rând, memoria consumată este destul de mare. De asemenea, fiind orientat pe proiecte, mediul de dezvoltare nu se pretează pentru programele mici (sub 300 de linii).

14.9.2 Anjuta

Spre deosebire de Eclipse, Anjuta[5] este un IDE pentru C și C++ pentru GNU/Linux. Scris pentru GTK și Gnome, programul oferă o serie de facilități precum managementul proiectelor, un depanator interactiv și un editor de cod ce permite syntax highlight și source browsing.

De asemenea, sunt posibile, print intermediul unor plugin-uri scrise în C (în viitor C++ și Python) diverse configurări. De exemplu, se poate schimba editorul de cod din Scintilla

[1] http://valgrind.org/docs/manual/manual.html
[2] http://valgrind.org/docs/manual/mc-manual.html#mc-manual.errormsgs
[3] http://valgrind.org/docs/manual/mc-manual.html#mc-manual.options
[4] http://www.eclipse.org
[5] http://projects.gnome.org/anjuta/index.shtml

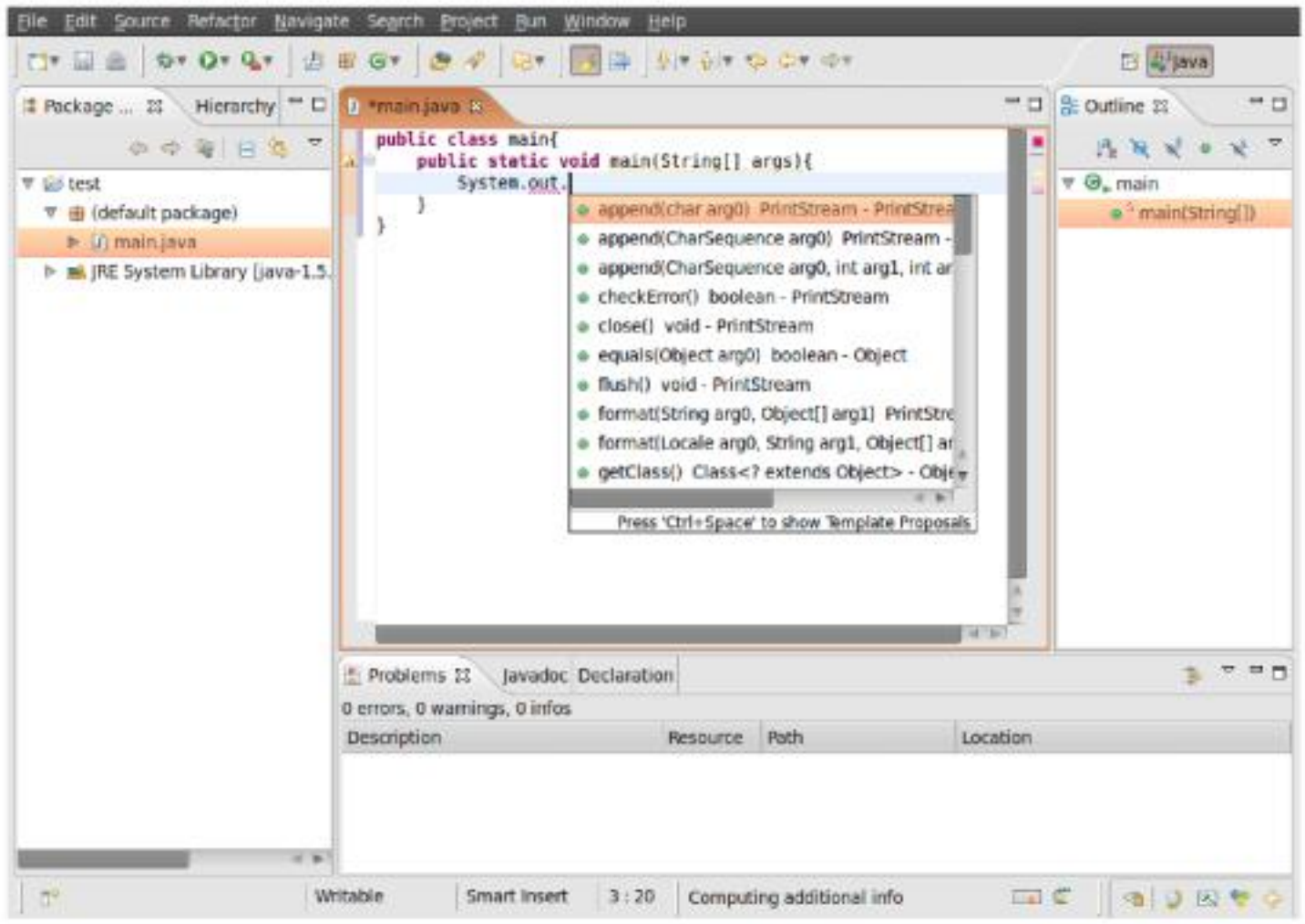


Figura 14.1: Eclipse

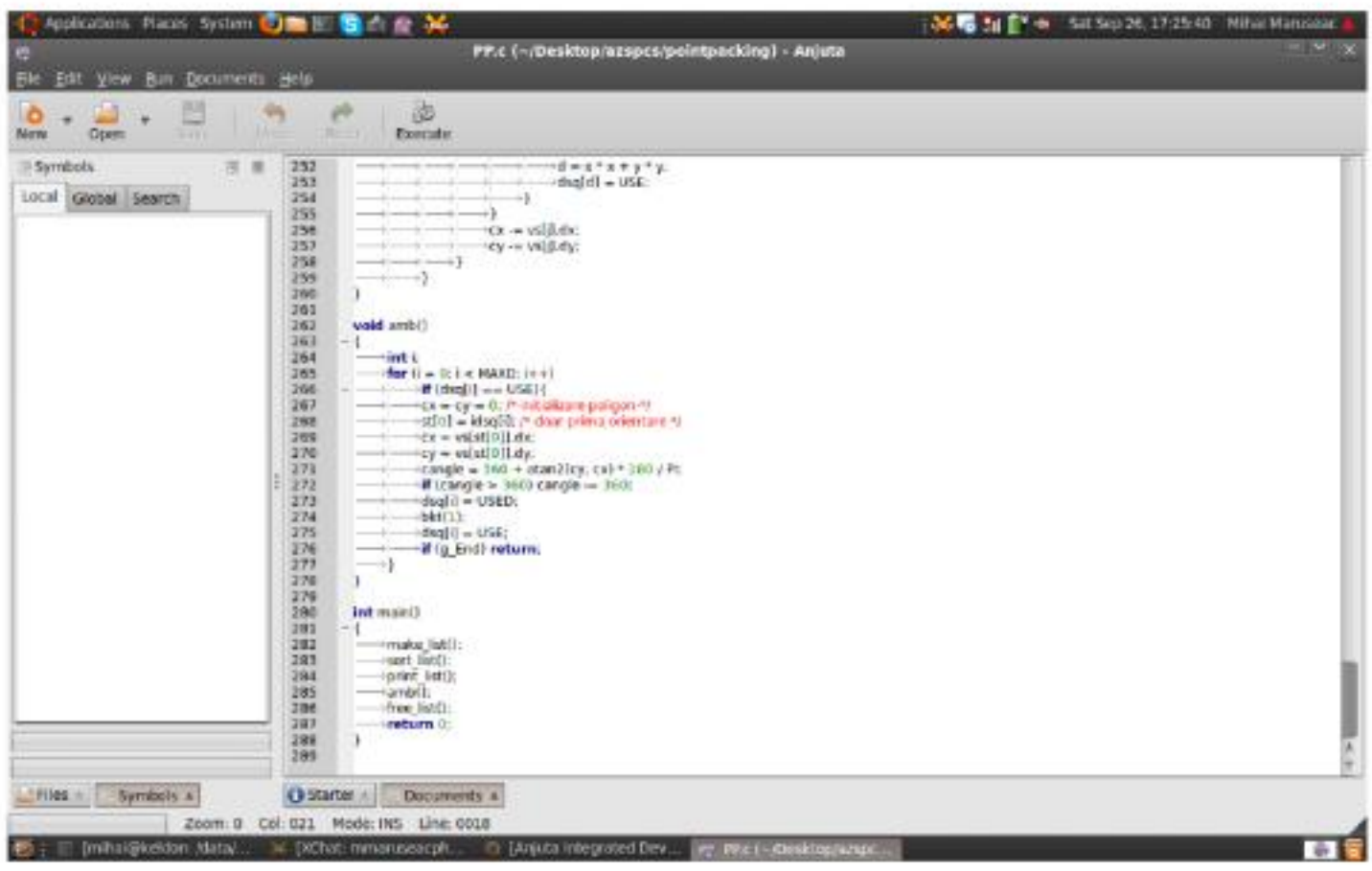


Figura 14.2: Anjuta

sau GtkSourceView în vim sau Emacs prin intermediul unui singur script scris de utilizator. Dacă Anjuta detectează mai multe plugin-uri pentru același scop, utilizatorul va fi rugat să selecteze unul din acestea, selecția fiindu-i memorată.

De asemenea, prin intermediul plugin-urilor se pot introduce facilități de source control (SVN). Mult mai important este faptul că managerul de proiecte nu reține informații despre proiect în alt fișier, astfel fiind posibilă dezvoltarea într-un grup mixt de utilizatori,

unii folosind Anjuta, alții folosind editoare clasice (vim, Emacs).

Pentru dezvoltarea aplicațiilor grafice Gtk, Anjuta oferă un plugin pentru interacțiunea cu Glade. Acceptând și limbajul C++, Anjuta oferă un generator de clase și un plugin ce va arăta ierarhia claselor din proiect. De asemenea, există plugin-uri pentru gdb, valgrind și gprof (ultimul pentru profiling-ul codului scris).

14.10 Managementul proiectelor software

O componentă importantă în dezvoltarea aplicațiilor software o reprezintă activitățile de gestiune, monitorizare și control a acestora. Managementul proiectelor software este legat de procesul de dezvoltare software și de modul în care o aplicație este proiectată, dezvoltată, testată și prezentată.

Activitățile de management al proiectelor software se referă la înțelegerea cerințelor aplicației, arhitectura și proiectarea acesteia, împărțirea de roluri și responsabilități în cadrul echipei, motivarea echipei, rezolvarea conflictelor, discuții, monitorizarea și controlul activităților, feedback, evaluarea rezultatelor, analiza calității. Domeniul managementului proiectelor sofware are o relevanță deosebită întrucât fiecare proiect software diferă de un altul. De asemenea, întrucât niciun limbaj de programare nu este perfect, procesele de dezvoltare trebuie să fie bine corelate cu limbajele, tehnologiile și framework-urile folosite.

14.10.1 Procese de dezvoltare software

Procesul de dezvoltare software reprezintă structura de activități folosită pentru dezvoltarea unui produs software.

Activitățile cuprinse în procesul de dezvoltare software sunt:

- **planificarea** – se referă extragerea cerințelor și stabilirea funcționalităților;
- **proiectarea** – se referă la stabilirea specificațiilor și a arhitecturii (module sau componente și interacțiunile între acestea);
- **implementare** – se referă la dezvoltarea efectivă a aplicației;
- **testarea** sau **verificarea** – însemnă verificarea respectării cerințelor software și a funcționării în condiții optime a aplicației;
- **documentarea** aplicației;
- **lansarea** aplicației pe piață sau livrarea acestuia clientului;
- **mentenanța** – se referă asigurarea funcționării corecte a aplicației în condițiile apariției de noi probleme și răspunsuri la solicitările clienților.

Există două tipuri de modele pentru implementarea procesului de dezvoltare software: modelul clasic (*waterfall*[1]) și modelul iterativ[2].

Modelul clasic presupune o dezvoltare pas cu pas a activităților procesului de management. O activitate nu va începe cât timp activitatea anterioară nu a fost definitivată. Acest model are avantajul unei dezvoltări riguroase, dar oferă o flexibilitate redusă, iar o echipă care lucrează la o anumită componentă va trebui să aștepte ca echipele anterioare să încheie componente proprie.

Modelul de dezvoltare iterativă presupune construirea unei părți reduse a proiectului software și apoi creșterea acesteia. Procesul iterativ este preferat în momentul în care nu se cunoaște foarte bine scopul urmărit. O subclasă a modelului iterativ o reprezintă dezvoltarea agilă (*agile software development*[3]). Dezvoltarea agilă folosește mecanismul de feedback ca principal mecanism pentru controlul și evoluția proiectului. Un alt exemplu de model iterativ este *Extreme Programming*[4] care presupune dezvoltarea de părți foarte mici ale unui proiect (de obicei în echipe mici) și apoi integrarea acestora în proiect.

14.10.2 Aplicații web pentru managementul proiectelor software

Există un set divers de aplicații de gestiune a proiectelor software. În cadrul acestora, o categorie aparte o reprezintă aplicațiile web pentru managementul proiectelor software. Aceste aplicații oferă, în general, un set integrat de componente printre care:

- wiki – un wiki este o aplicație web care permite editarea colaborativă facilă de conținut; celebra enciclopedie online Wikipedia folosește tehnologia wiki pentru dezvoltare colaborativă;
- un repository (vezi secțiunea 14.4) pentru gestiunea codului sursă;
- sistem de tichete pentru raportarea problemelor (*bug-urilor*) și pentru cerințe de noi funcțialități;
- roadmap și milestone-uri pentru planificarea și contorizarea activităților.

Trac[5] este o aplicației minimalistă pentru gestiunea proiectelor software. Trac folosește Subversion ca aplicație pentru gestiunea codului. Un aspect pozitiv al Trac este numărul mare de plugin-uri disponibile[6]. Trac este scris în Python și este folosit de un număr mare de organizații. **Redmine**[7] este o aplicație web pentru gestiunea proiectelor software scrisă folosind Ruby on Rails[8]. Redmine a fost influențat puternic de Trac. Redmine oferă în plus față de Trac o diversitate de utilitare de gestiune a codului (Subversion, Git, Mercurial, Bazaar, Darcs), gestiunea de proiecte multiple în cadrul aceleiași instanțe și folosirea de diagrame Gantt și de calendare.

[1] http://en.wikipedia.org/wiki/Waterfall_model
[2] http://en.wikipedia.org/wiki/Iterative_development
[3] http://en.wikipedia.org/wiki/Agile_software_development
[4] http://en.wikipedia.org/wiki/Extreme_Programming
[5] http://trac.edgewall.org/
[6] http://trac-hacks.org/
[7] http://www.redmine.org/
[8] http://www.rubyonrails.org/

14.10.3 Resurse online pentru dezvoltarea proiectelor

Pentru cei care nu doresc instalarea unei instanțe de Trac sau Redmine pentru dezvoltarea și managementul proiectelor sofware, se po folosi resurse online. În general, accesul la acestea este gratis. Oricine poate crea un cont asociat unui proiect și poate folosi resursele puse la dispoziție pentru dezvoltarea acestuia.

Aplicațiipe ce stau la baza acestor site-uri sunt denumite `software forge-uri`[1]. Majoritatea forge-urilor pot fi instalate separat. Principalele site-uri care oferă servicii de găzduire a proiectelor software sunt SourceForge[2], GNU Savannah[3], Google code[4], BerliOS[5], CodePlex[6].

14.11 Resurse de documentare pentru dezvoltator

Mai mult decât în orice alt domeniu, în IT documentația este ușor de obținut, de la format electronic până la cărți tradiționale, din surse oficiale sau contribuții ale altor persoane.

14.11.1 Documentație oficială

Pachetele software cunoscute vin, în general, cu documentație detaliată. Aceasta este accesibilă, în cazul programelor simple și a bibliotecilor, prin pagini de manual (**`man`**).

În cazul programelor GNU[7], manualul nu conține decât un sumar al comenzilor. Pentru informații mai detaliate se folosește **`info`**. Există alternative la **`info`**, printre care cititorul integrat din Emacs, sau **`pinfo`**[8].

Alte suite software complexe, în general limbaje de programare, posedă utilitare personalizate pentru vizualizarea documentației. Câteva exemple sunt:

- `pythondoc` pentru documentarea limbajului Python[9];
- `perldoc` pentru documentarea limbajului Perl[10];
- `texdoc` pentru documentarea limbajului TEX[11] și a suitei de macro-uri pentru el, LATEX[12].

[1] http://en.wikipedia.org/wiki/Forge_(software)
[2] http://sourceforge.net/
[3] http://savannah.gnu.org/
[4] http://code.google.com/
[5] http://www.berlios.de/
[6] http://www.codeplex.com/
[7] http://www.gnu.org/
[8] http://pinfo.sourceforge.net/
[9] http://www.python.org/
[10] http://www.perl.org
[11] http://tug.org/
[12] http://www.latex-project.org/

14.11.2 Documentarea programelor proprii

Orice program trebuie documentat, chiar dacă nu este intenția explicită a programatorului ca sursa să fie citită de altcineva. Deși, în general, comentarii sumare la începutul fiecărei funcții sunt suficiente, există programe care permit comentarea structurată și generarea de documentație în varii formate (HTML, pagini de manual, LaTeX). Este, însă, important de menționat că documentația generată este adresată *dezvoltatorilor*, și nu utilizatorilor.

Pentru ca un astfel de program să funcționeze corect, este nevoie de o metodă de a delimita comentariile uzuale de cele care contribuie la generarea automată a documentației. Două exemple de formate speciale pentru limbajele din familia C/C++ sunt începerea comentariilor cu `/**` sau cu `///`.

Un astfel de program este **Doxygen**[1]. Este open-source, are suport pentru mai multe limbaje și generează documentație în formate variate. Este, în general, prima alegere pentru proiecte open-source care aleg să genereze documentația automat. Un exemplu excelent de documentație generată de Doxygen este cea a proiectului Pidgin[2].

Un exemplu de comentariu compatibil cu Doxygen, pentru o funcție C:

```
1  /**
2   * Executa comenzile corespunzatoare unui arbore.
3   *
4   * @param c comanda compusa care trebuie executata
5   * @param pi descriptorul care trebuie duplicat pentru intrare
6   * @param po descriptorul care trebuie duplicat pentru iesire
7   * @param w este 1 daca trebuie sa asteptam comanda sa se termine
8   *
9   * @return Codul de iesire al ultimului proces executat, sau pidul
10  * procesului nou creat daca acesta nu a fost asteptat. Daca comanda
11  * executata a fost quit sau exit, se intoarce valoarea speciala
12  * QUITTING.
13  *
14  * @remarks pi si po sunt folosite atunci cand comanda face parte
15  * dintr-un pipe. Pentru a nu le folosi, le setam la -1.
16  */
17 int exec_command(const command_t *c, int pi, int po, int w)
18 {
19   [...]
```

Limbajul Java are un instrument propriu pentru generarea documentației de API, numit `Javadoc`[3]. Toată documentația platformei Java[4] este generată utilizând `Javadoc`, iar formatul Doxygen este voit compatibil cu cel Javadoc pentru a fi o alternativă facilă.

C# oferă o alternativă interesantă, folosind comentarii în format XML și marcate cu `///`. Deși marcarea câmpurilor este mai dificilă, mediul Visual Studio oferă suport excelent pentru acest tip de comentarii.

---

[1] http://www.doxygen.org/
[2] http://developer.pidgin.im/doxygen/dev/html/classes.html
[3] http://java.sun.com/j2se/javadoc/
[4] http://java.sun.com/j2se/1.4.2/docs/api/

14.11.3 Cărți și tutoriale

Un tutorial este o "scurtătură", o referință rapidă, dar incompletă a unei tehnologii sau limbaj. O astfel de resursă poate substitui temporar documentația propriu-zisă, în faza de familiarizare, dar detaliile care lipsesc (intenționat, pentru a asimila mai ușor lucrurile de bază) sunt adesea esențiale pentru o utilizare eficientă a conceptului în cauză.

Deși documentația "la obiect" este accesibilă sub formă electronică, știința calculatoarelor nu duce lipsă de cărți în format clasic. Acestea nu sunt nicidecum redundante, pentru că deseori conțin sugestii, subtilități și practici foarte valoroase.

O categorie importantă de cărți sunt cele care își propun să detalieze un domeniu întreg, independent de implementare. Acestea conțin elemente teoretice avansate, dar și studii de caz relevante. În general, fiecare domeniu are o asemenea carte, recunoscută și respectată de persoanele în temă. Pentru că nu sunt "încuiate" în studiul unei implementări particulare, aceste cărți sunt preferate în domeniul academic.

Exemple de asemenea cărți și domeniile[1] lor asociate sunt:

- sisteme de operare: Silberschatz, Galvin, Gagne, *Operating System Concepts* [26]; Tanenbaum – *Modern Operating Systems* [32];
- rețele de calculatoare: Tanenbaum – *Computer Networks* [30];
- arhitectura sistemelor: Hennessy, Patterson – *Computer Architecture: A Quantitative Approach* [9];
- compilatoare: Aho, Sethi, Ullman – *Compilers: Principles, Techniques, and Tools* [2].

Nu în ultimul rând, profesorul Donald Knuth scrie o veritabilă monografie a programării calculatoarelor, numită *The Art of Computer Programming* [14]. Începută în 1962, cartea este programată să aibă 7 volume, dintre care doar primele 3 au apărut complet.

Alte cărți se referă exclusiv la programare și descriu în detaliu fie un limbaj, fie un set de utilitare sau un mediu întreg de programare. Spre deosebire de categoria anterioară, aceste cărți se adresează unui domeniu mai îngust de probleme, dar adoptă o abordare *hands-on*, conținând frecvent exemple de cod.

Lucrări celebre în acest sens sunt: Kernighan, Ritchie – *The C Programming Language* [13], Wall, Christiansen, Orwant – *Programming Perl*[2] [33], Kernighan, Pike – *The Practice of Programming* [12], sau Eckel – *Thinking in Java* [5].

În cele din urmă, există cărți care descriu în detaliu un program sau o parte a unui sistem complex. Acestea se remarcă printr-un grad înalt de specificitate și limbaj tehnic, dar oferă o detalii fine despre subiectul abordat. Asemenea lucrări sunt apreciate în special în industrie, de persoane care configurează sau modifică programe ca parte a slujbei lor.

În această categorie se încadrează, de exemplu, cărțile lui Robert Love despre kernelul Linux, *Learning the vi and Vim Editors* de Robbins, Hannah și Lamb, sau *DNS and BIND* de Liu și Albitz.

---

[1] Desigur, domeniile se întrepătrund. Un inginer nu își permite să se limiteze la un singur aspect al profesiei lui.

[2] cunoscută cu afecțiune în comunitatea Perl drept *The Camel Book*

Departe de domeniul tehnic, dar relevante pentru industria IT, sunt lucrările cu tentă ideologică sau cele care observă și compară tendințe. Acestea tind să se refere la procesul producerii software-ului, fie el proprietar sau liber. Titluri notabile sunt Brooks – *The Mythical Man-Month*, Raymond – *The Cathedral and the Bazaar* și Fogel – *Producing Open Source Software*[1].

Ca o notă finală asupra cărților, este esențial a realiza că ele dezvoltă competențe distincte față de citirea documentației exclusiv tehnice. Cele două tipuri de resurse se folosesc împreună – ele se completează, nu se substituie.

14.11.4 Documentație din Internet

Pe lângă documentația oficială, există multe alte surse, fiecare cu specificul ei. Gradul de detaliu și interacțiunea dintre persoane variază funcție de mijlocul de comunicație.

IRC

IRC (*Internet Relay Chat*) este un protocol de comunicație multidirecțional în timp real. Clienți uzuali pentru acest mediu sunt Pidgin[2], XChat[3], sau irssi[4].

Deși protocolul însuși, în special datorită utilizării programului mIRC, are o reputație îndoielnică, există mult pachete software și comunități care oferă suport informal folosind IRC. Este utilă o asemenea abordare când se dorește un răspuns în timp real, la obiect, dar din care de multe ori lipsesc detalii care nu sunt imediat relevante.

Majoritatea proiectelor care folosesc IRC drept mijloc de comunicare sunt găzduite de rețeaua Freenode[5]. Dezvoltatorii se lansează frecvent în discuții informale, fără legătură cu subiectul, de aceea protocolul are un *signal-to-noise ratio* destul de mic.

Usenet

Usenet este o rețea care precede cronologic World Wide Web-ul. Canalul de comunicație este tot *one-to-many*, dar, spre deosebire de IRC, această metodă este asincronă, asemănându-se cu email-ul.

Datorită vârstei "venerabile" a și a lipsei sale relative de popularitate, Usenet este folosit de unii utilizatori mai în vârstă, dar care au cunoștințe tehnice excepționale. Usenet conține discuții tehnice detaliate, dar și atacuri personale și *flame-wars*.

În ultimii ani, ISP-urile au încetat să mai ofere servere de Usenet, dar o metodă excelentă de a citi și scrie mesaje este arhiva Google Groups[6]. Un grup de o notorietate legendară, atât pentru discuțiile referitoare la limbajul C, cât și pentru flame-urile numeroase, este `comp.lang.c`.

---

[1] http://producingoss.com/
[2] http://pidgin.im/
[3] http://xchat.org
[4] http://irssi.org
[5] http://freenode.net/
[6] http://groups.google.com/groups/dir?sel=gtype%3D0

Mai mult decât orice altă sursă de informație, Usenet este neiertător cu persoanele care fac greșeli. Se recomandă citirea câtorva tread-uri de discuție înainte de a porni o discuție nouă[1].

Google

O ultimă sursă de informație este a folosi un motor de căutare. Avantajul este viteza cu care acesta caută prin majoritatea surselor menționate mai sus, inclusiv discuții Usenet sau log-urile canalelor de IRC. Pe de altă parte, este uneori dificilă alegerea cuvintelor potrivite pentru exprimarea unei probleme.

14.12 Studiu de caz

14.12.1 Microsoft Visual Studio

Microsoft Visual C++ (MSVC) este un IDE(Integrated Development Enviroment) făcut de Microsoft pentru limbajele de programare C, C++ și C++/CLI. Acesta conține utilitare de depanare și dezvoltare a codului scris în C++.

Visual C++, componenta care se ocupă de codul C/C++, este compus din mai multe elemente:

- compilatorul Visual C++, conține optimizări pentru platformele x86, x64 și Itanium
- bibliotecile Visual C++, cum ar fi ATL(Active Template Library), MFC(Microsoft Foundation Class), CRT(C RunTime Library) și biblioteca standard C
- mediul de dezvoltare Visual C++. Compilatorul și bibliotecile pot fi accesate și prin interfața în linie de comandă, mediul de dezvoltare oferă posibilități de dezvoltare și depanare mai facilă a proiectelor

Există mai multe metode de a crea un proiect în Visual C++:

- se poate folosi un șablon, cum ar fi *Console Application Template*, pentru a realiza proiecte simple
- se poate folosi și un wizard pentru crearea unei soluții, o soluție poate să conțină mai multe proiecte
- se poate crea un fișier text simplu salvat cu extensia *.cpp*, iar acesta să fie inclus într-un proiect gol de tipul *Win32 Application*

Folosind wizard-ul avem acces la o interfață ce permite să se creeze un proiect, să se modeleze un proiect după un șablon și să se generezeze fișiere sursă și directoare pentru aplicație. Visual C++ lucreaza cu framework-uri de aplicații și biblioteci pentru a crea schelete de programe peste care se poate lucra ulterior.

Mai multe detalii despre proiecte puteți găsi pe site-ul MSDN[2].

---

[1] *Lurk before you leap.*

[2] http://msdn.microsoft.com/en-us/library/6765tta0.aspx

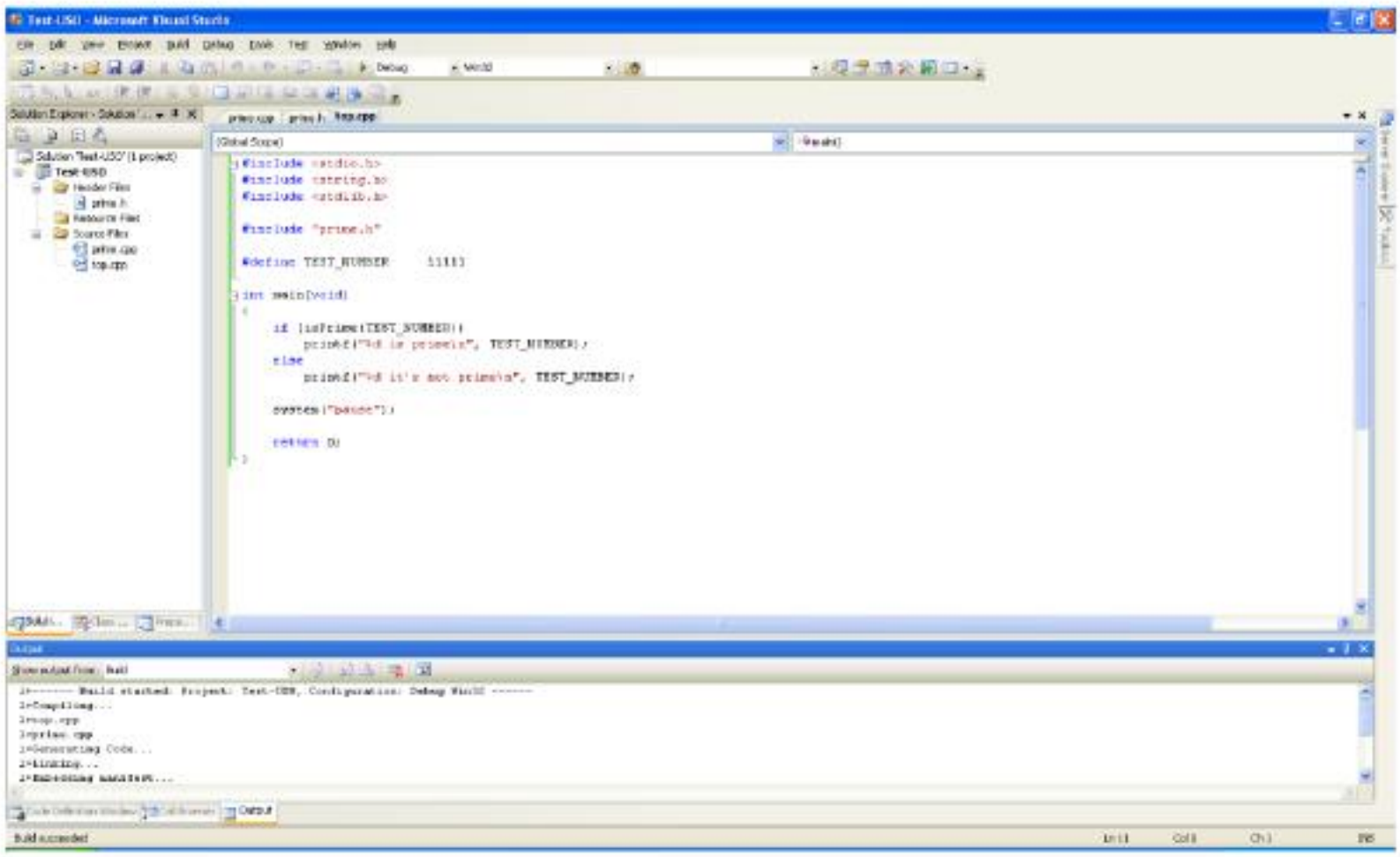


Figura 14.3: Proiect în Visual C++

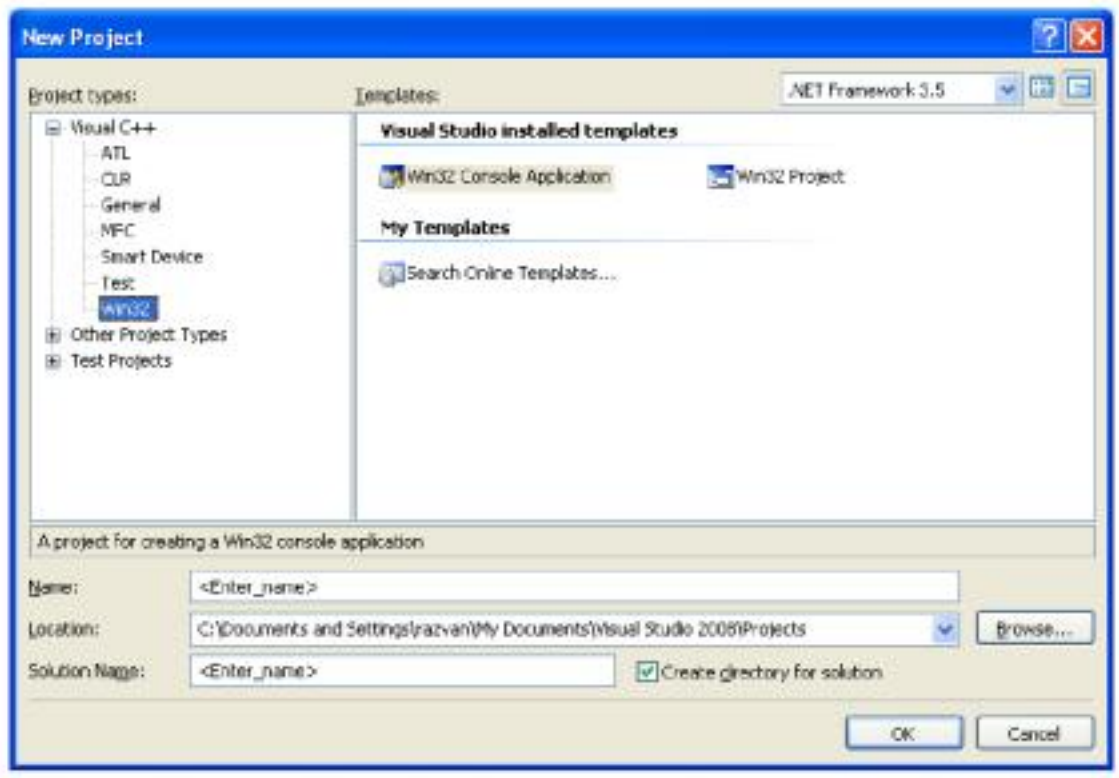


Figura 14.4: Creare proiect în Visual C++

Cuvinte cheie

- coding style
- indent, astyle
- editor
- vim
- modul insert, command, visual
- controlul versiunii
- subversion
- git
- ctags
- fișiere index (tag)
- analiză statică
- splint

- GNU Autotools
- strace
- .data, .bss, .rodata
- stivă
- heap
- apeluri de sistem
- ldd
- depanare
- GDB
- ddd
- Valgrind
- memory leak
- IDE
- Eclipse
- Anjuta
- waterfall
- model iterativ
- Trac
- Redmine
- man
- info
- Doxygen
- Javadoc
- IRC – Internet Relay Chat
- Usenet
- Microsoft Visual Studio

Întrebări

1. Care din următoarele este un editor?
   - ❑ Vim
   - ❑ astyle
   - ❑ ctags
   - ❑ gdb
2. Care din următoarele aplicații este folosită pentru depanarea la runtime a unui program?
   - ❑ Valgrind
   - ❑ Trac
   - ❑ indent
   - ❑ git
3. Care din următoarele operații NU este o operație validă la folosirea unui sistem de gestionare a versiunii?
   - ❑ commit
   - ❑ update
   - ❑ checkout/clone
   - ❑ boot
4. Care din următoarele NU se referă la recomandări din cadrul unor convenții de codare?

- ❑ nume sugestive pentru variabile
- ❑ funcții cu dimensiune rezonabilă
- ❑ folosirea de spații și linii libere
- ❑ licențierea codului sub GPL

5. Care din următoarele NU este un IDE?
   - ❑ Anjuta
   - ❑ Eclipse
   - ❑ Microsoft Visual Studio
   - ❑ OpenOffice
6. Care din următoarele utilitare este folosit în conjuncție cu Vim pentru parcurgerea facilă a codului?
   - ❑ ctags
   - ❑ splint
   - ❑ indent
   - ❑ gdb
7. Care din următoarele NU este un mod de folosire pentru editorul Vim?
   - ❑ `visual`
   - ❑ `command`
   - ❑ `insert`
   - ❑ `graphic`
8. Care din următoarele comenzi este folosită, în Vim, pentru salvarea unui fișier?
   - ❑ `:w`
   - ❑ `:q`
   - ❑ `:s`
   - ❑ `:help`
9. Care din următoarele este un sistem pentru controlul versiunii?
   - ❑ Git
   - ❑ Vim
   - ❑ Eclipse
   - ❑ ddd
10. Care din următoarele este o aplicație web pentru gestiunea proiectelor software?
    - ❑ Trac
    - ❑ GTK

- ddd
- Waterfall

Capitolul 15

Viața în Linux

See, you not only have to be a good coder to create a system like Linux, you have to be a sneaky bastard too ;-)

Linus Torvalds

Ce se învață din acest capitol?

- Cu ce se ascultă muzică în Linux
- Cum se vede un film în Linux
- Cum se scrie un CD/DVD în Linux
- Cum se foloseste Y!M în Linux
- Cum se descarcă conținut BitTorrent în Linux
- Cum se folosește imprimanta în Linux
- Ce jocuri 3D și 2D există în Linux
- Cum se folosesc două monitoare în Linux
- Cum se editează documente Word/Excel în Linux

15.1 Muzica în Linux

Un player bun de muzică trebuie să asigure, pe lângă posibilitatea de a reda fișierele audio, managementul acestora, salvarea playlist-urilor, posibilitatea de "minimize to tray" și, eventual, posibilitatea de audio-scrobbling (trimiterea informațiilor despre muzica ascultată unor site-uri de profil, precum last.fm[1]). Un utilizator mai pretențios ar cere și management-ul fișierelor de pe Audio-CD-uri, iPod-uri etc.

Din fericire, majoritatea acestor condiții sunt respectate de player-ele audio din Linux, fără a fi necesară instalarea unor plugin-uri suplimentare (acestea fiind instalate automat în momentul instalării aplicației).

[1] http://www.last.fm/

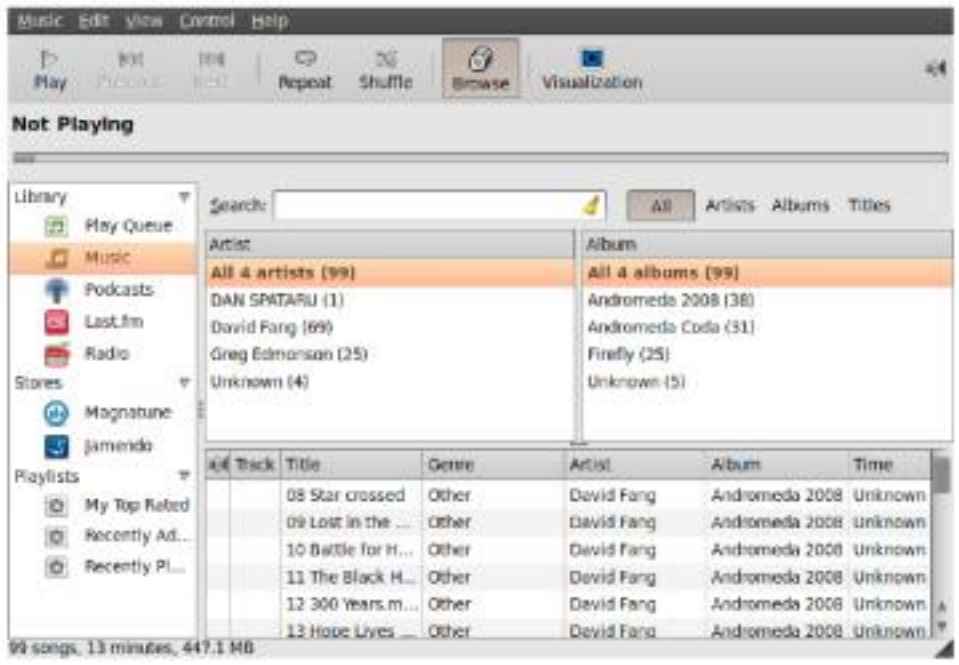


Figura 15.1: Rhythmbox

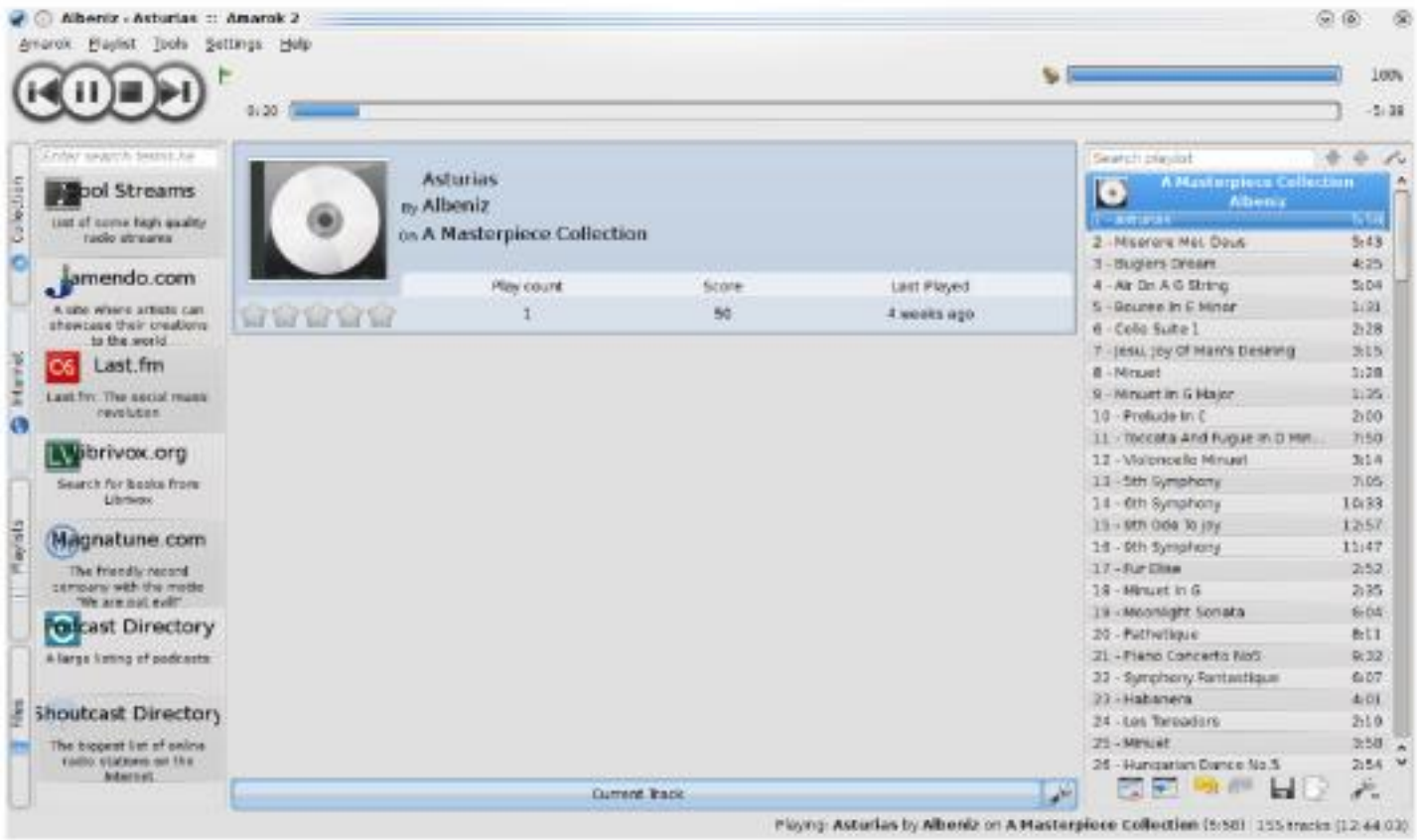


Figura 15.2: Amarok

Sistemele de operare Ubuntu au preinstalat **Rhythmbox**. Nu este doar un player audio ci și o aplicatie pentru managementul fișierelor audio, inspirată din aplicația iTunes de la Apple. Echivalent, în mediile Kubuntu este preinstalat **Amarok**.

Dacă aplicațiile preinstalate nu corespund preferințelor lui, utilizatorul poate oricând să instaleze altele, precum:

- **Banshee**[1] – o aplicație similară Rhythmbox, bazată pe mono
- **Listen**[2] – o aplicație minimală scrisă în Python
- **Audacious**[3] – un player similar aplicației Winamp din Windows

[1] http://banshee-project.org/
[2] http://www.listen-project.org/
[3] http://audacious-media-player.org

Figura 15.3: Audacious

- **Exaile**[1] – o aplicație similară Amarok pentru GNOME

O aplicație interesantă este Music Player Daemon[2]. Așa cum îi spune și numele, programul rulează ca daemon, dar este controlat, fie local, fie prin rețea, de un client. Există zeci de astfel de clienți, pentru diverse platforme, scriși în varii limbaje de programare și cu interfețe grafice sau în mod text.

15.2 Filme în Linux

Vizualizarea de filme este deseori mai facilă în Linux decât în Windows, pentru că programele care există sunt toate gratuite în adevăratul sens al cuvântului, adică nu vin cu alt software de proveniență îndoielnică. Mai mult, în general instalarea unui video player pe Linux implică automat instalarea codecurilor asociate, dacă ele nu sunt deja instalate.

VLC este un player produs de proiectul VideoLAN. Acesta are avantajul de a avea codec-uri integrate, ceaa ce reduce pachetele adiționale care trebuiesc instalate cu el. Pe sistemele Debian, pachetul asociat se numește `vlc`.

O alternativă la VLC este `mplayer`, cu o interfață mai minimală, dar la fel de bogat în facilități. Utilizatorii `mplayer` pot alege între mai multe variante de interfețe, de la mod text până la GUI-uri sofisticate și interfețe native (pentru Mac OS X, de exemplu). `mplayer` are, de asemenea, posibilitatea de a reda filme folosind mai multe drivere de output, folosind parametrul `-vo` în linie de comandă. Astfel, dacă pachetele potrivite sunt instalate, un utilizator nu are, de exemplu, nevoie de interfață grafică pentru a vizualiza un fișier. `mplayer` poate afișa direct în framebuffer folosind biblioteca `directfb` sau în mod text folosind `aalib`[3].

Ambele programe menționate folosesc preponderent biblioteca `libavcodec`, parte a proiectului FFmpeg. Pentru unele formate specifice Windows, se pot instala drivere

[1] http://www.exaile.org

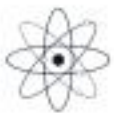
[2] http://mpd.wikia.com/wiki/Music_Player_Daemon_Wiki

[3] `aalib` folosește text pentru a reda imagini cât mai bine, deci poate fi folosit peste conexiuni text-only.

proprietare, numite generic `w32codecs`. Nu în ultimul rând, ambele pachete sunt portabile, deci pot fi rulate cu ușurință și pe Windows.

15.3 Scrierea unui CD/DVD în Linux

În mod clasic, scrierea de CD-uri/DVD-uri în Linux se realizează cu ajutorul utilitarului **cdrecord**. În ultimele versiuni, aplicația a fost denumită **wodim**. Aplicațiile grafice folosite uzual pentru scrierea de CD-uri (GnomeBaker, Brasero, K3b) folosesc în spate utilitarul **cdrecord**.



Pentru a folosi **wodim**/**cdrecord** este nevoie de o imagine de CD (fișier `.iso`. Pentru a obține o astfel de imagine se folosește utilitarul **`genisoimage`** (fostul **`mkisofs`**):

```
1  razvan@valhalla:~/projects$ ls library/
2  eLiberatica_OReilly_books.odt  eLiberatica_OReilly_lista_carti.gnumeric
3  eLiberatica_OReilly_carti.odt  eLiberatica_OReilly_lista_carti.ods
4
5  razvan@valhalla:~/projects$ genisoimage -o library.iso library/
6  Using ELIBE000.ODT;1 for  /eLiberatica_OReilly_carti.odt (
   eLiberatica_OReilly_books.odt)
7  Total translation table size: 0
8  [...]
9
10 razvan@valhalla:~/projects$ file library.iso
11 library.iso: ISO 9660 CD-ROM filesystem data 'CDROM
```

Scrierea CD-ului se face cu o comandă ca cea de mai jos:

```
1  razvan@valhalla:~/projects$ cdrecord dev=/dev/hda speed=8 library.iso
```

Imaginea `.iso` poate fi montată și accesată în sistemul local de fișiere, similar utilitarului Daemon Tools[1] din Windows:

```
1  razvan@valhalla:~/projects$ mkdir tmp_mount
2
3  razvan@valhalla:~/projects$ sudo mount -o loop -t iso9660 library.iso
   tmp_mount/
4
5  razvan@valhalla:~/projects$ ls tmp_mount/
6  elibe000.odt  eliberat.gnu  eliberat.ods  eliberat.odt
```

Se observă că, în urma obținerii imaginii, fișierele au fost redenumite în formatul **8.3**[2].

În GNOME, scrierea CD-urilor se poate realiza cu ajutorul utilitarului **GnomeBaker**. Pentru scrierea unui CD, se pornește utilitarul, se selectează opțiunea `Data CD` și apoi se adaugă directoarele de scris pe CD în arborescența din partea stânga jos, așa cum reiese din figura 15.4. În momentul în care toate fișierele au fost adăugate se folosește butonul `Burn`.

În KDE, scrierea CD-urile se realizează cu ajutorul utilitarului **K3b** (vezi figura 15.5). Modul de utilizare este foarte apropiat de cel al GnomeBaker.

[1] http://www.daemon-tools.cc/eng/home
[2] http://en.wikipedia.org/wiki/8.3_filename

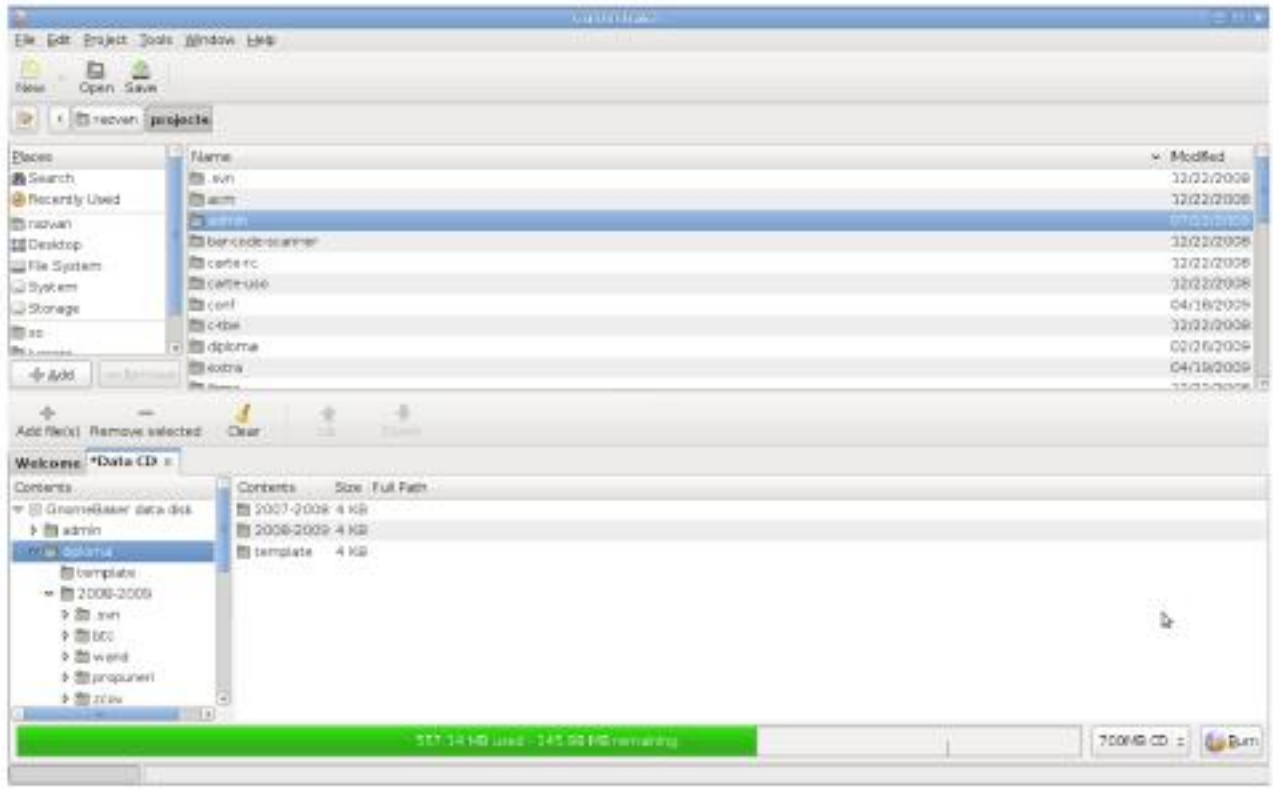


Figura 15.4: Scrierea unui CD folosind GnomeBaker

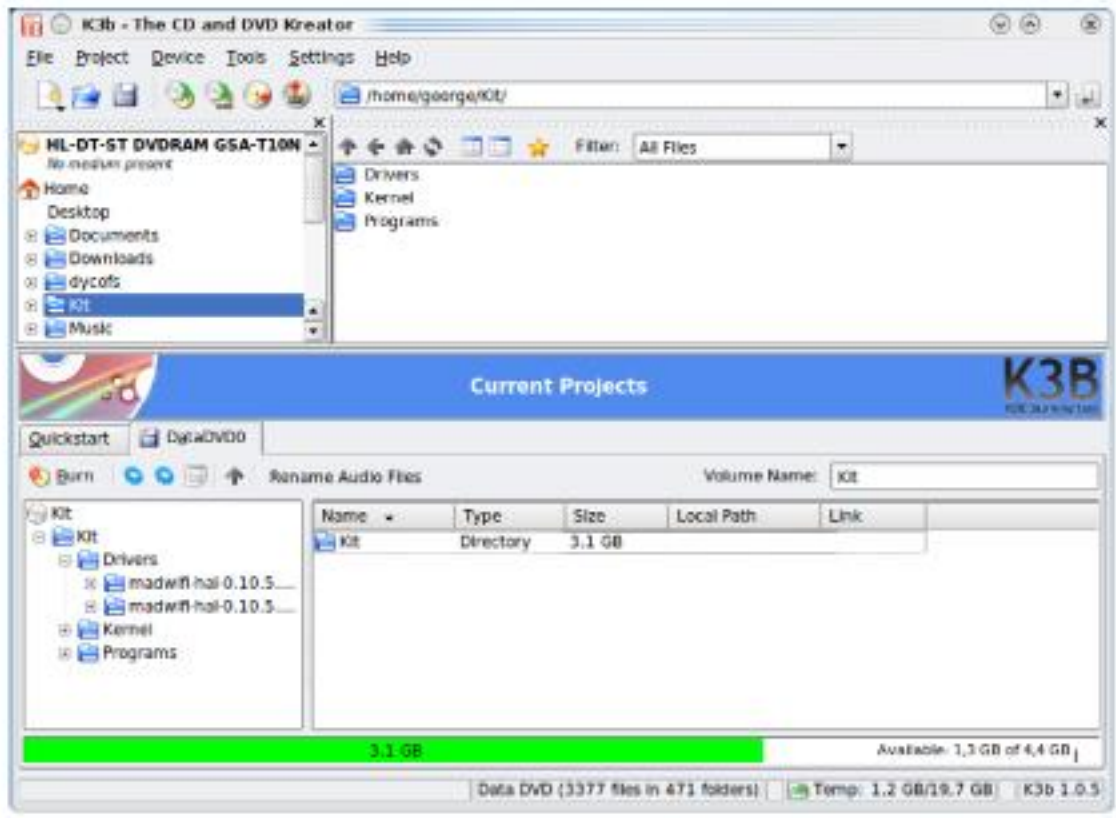


Figura 15.5: Scrierea unui CD folosind K3b

15.4 Messenger în Linux

Pentru utilizatorii de *Instant Messaging*, Linux are, tradițional, o alternativă ușor diferită, dar mult mai flexibilă și mai utilizabilă.

Acest program se numește **Pidgin**. Inițial numit Gaim, Pidgin are avantajul de a fi multi-protocol. Folosind aceeași interfață și o singură instanță a programului, un utilizator poate avea acces la conturile de Yahoo Messenger, MSN/Windows Live, Google Talk, AIM etc. Mai mult, este posibilă folosirea mai multor conturi din aceeași rețea simultan, fără a fi nevoie de instalarea unui patch adițional.

Central conceptului de conturi multiple din Pidgin este fereastra "*Accounts*". Aceasta poate fi accesată din meniul `Accounts→Manage Accounts`. Adăugarea unui cont

de Yahoo este cât se poate de facilă; ea nu necestă configurări adiționale. Google Talk necesită specificarea unui domeniu, pentru că se bazează pe un protocol descris numit XMPP[1]. Un exemplu de adăugare a unui cont Google Talk este prezentat în figura 15.6.

Figura 15.6: Adăugarea unui cont de Google Talk în Pidgin

Lista de contacte este una singură, pentru toate conturile active. Aceasta facilitează și gruparea mai multor contacte într-unul singur, pentru cazul în care aceeași persoană este prezentă în lista de contacte pe mai multe conturi sau rețele.

Pidgin are inclusiv suport de chat multidirecțional, cum ar fi conferințe de Yahoo sau protocolul IRC. Aceste *chat rooms* pot fi adăugate în lista de contacte, astfel programul are o interfață uniformă.

Nucleul Pidgin este biblioteca Purple. Aceasta mai este folosită, printre altele, și de Finch, un program cu interfață text-mode, dar asemănător cu Pidgin. Cele două programe și biblioteca asociată sunt dezvoltate de echipe cu legături foarte strânse și au pornit din același cod.

Alternative la Pidgin sunt Kopete (parte a mediului KDE), Empathy (care va înlocui Pidgin în unele distribuții), sau, cu interfață text, `centerim`.

15.5 BitTorrent în Linux

Pentru protocolul BitTorrent (protocol peer-to-perr de transfer de fișiere) există mai multe aplicații în Linux. Astfel, Ubuntu vine cu **Transmission**[2] preinstalat. Simplu și ușor configurabil, programul se remarcă prin faptul că utilizatorul nu trebuie să seteze nicio preferință inițial înainte de rularea acestuia (existând definite opțiuni by-default pentru ca toate lucrurile să funcționeze perfect). Desigur, opțiunile pot fi reconfigurate și acest

[1] http://xmpp.org
[2] http://www.transmissionbt.com/

lucru se face foarte ușor din interfața grafică, nefiind necesară o experiență foarte vastă. De asemenea, clientul Transmission are cel mai mic consum de memorie dintre toate aplicațiile ce folosesc protocolul BitTorrent.

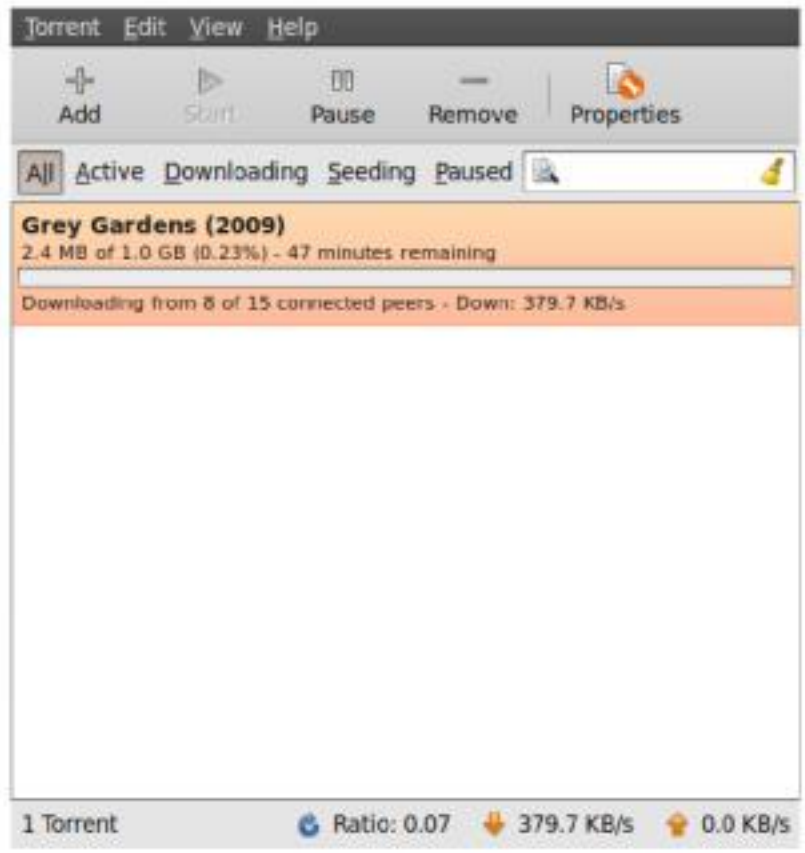


Figura 15.7: Transmission

Pentru Kubuntu există preinstalat **Ktorrent**[1]. Ca și în cazul Transmission, folosirea acestui program este foarte facilă, majoritatea configurărilor realizându-se prin interfața grafică.

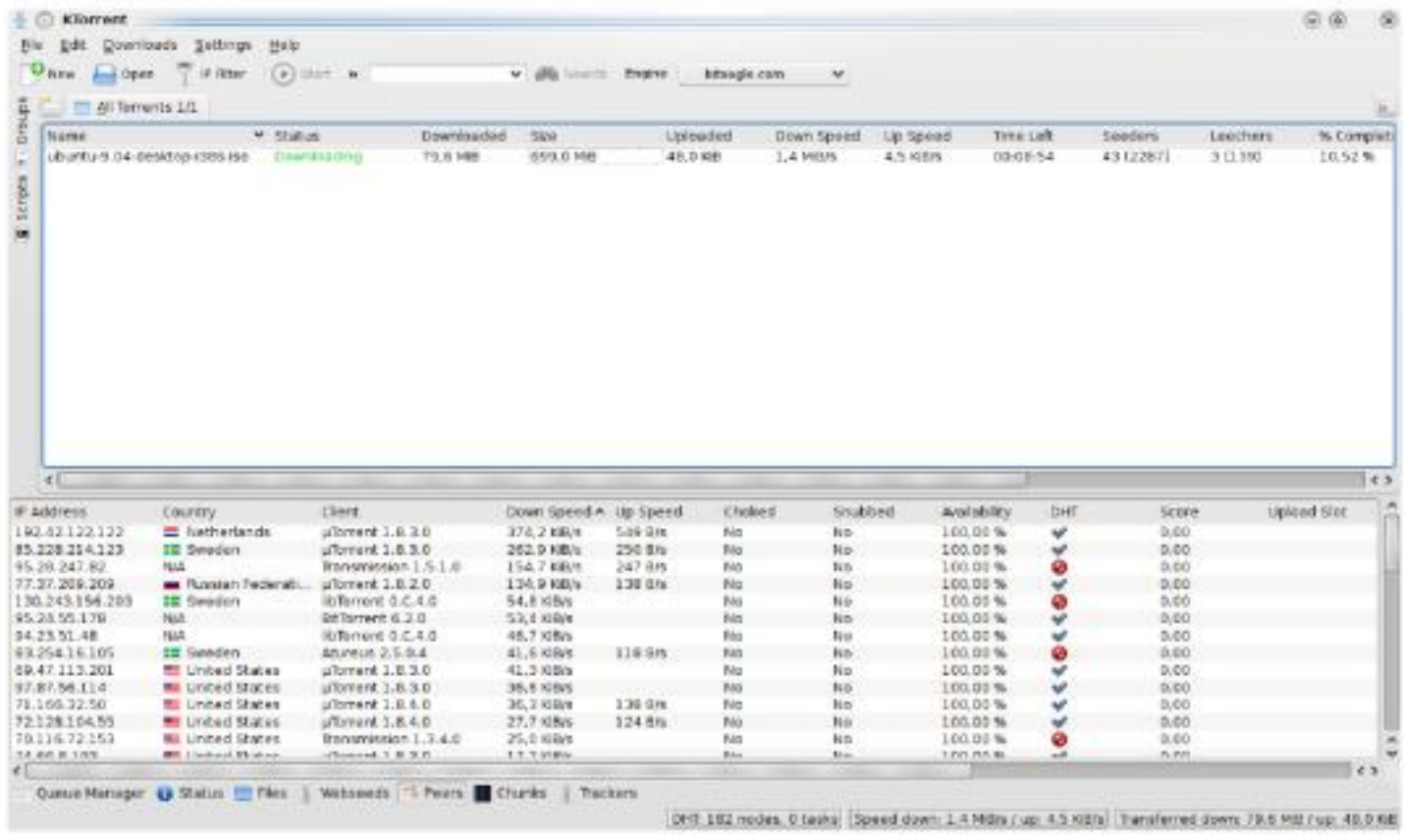


Figura 15.8: Ktorrent

Pentru cazul în care nu este disponibilă o interfață text, există posibilitatea folosirii

[1] http://ktorrent.org/

rtorrent[1]. Utilizând ncurses[2], această aplicație este foarte ușor de folosit peste o conexiune ssh sau într-un mediu fără interfață grafică.

Nu în ultimul rând, există **Limewire**[3], o aplicație cross-platform bazată pe Java, suportând facilități reduse de BitTorrent. În plus, aplicația folosește rețeaua Gnutella[4] pentru a localiza fișierele căutate.

15.6 Imprimanta în Linux

În Linux, folosirea imprimantei este condiționată de instalarea și configurarea **CUPS** (*Commmon Unix Printing System*). CUPS permite instalarea unui server care trimite solicitările către o imprimantă. Poate fi astfel folosit pentru a servi solicitări sosite din rețea. Pachetul asociat se numește `cups`.

În cadrul CUPS există pachetul `cups-pdf` care permite instalarea unei imprimante virtuale. Folosirea acestei imprimante virtuale conduce la obținerea unui fișier **PDF**[5]. În mod implicit, fișierele generate sunt stocate în directorul `PDF` din home-ul utilizatorului curent.

Imprimantele (sau serviciul CUPS) pot fi configurate în două moduri:

- prin intermediul interfeței grafice, în Gnome (vezi figura 15.9) sau KDE (vezi figura 15.10);
- prin intermediul interfeței web (vezi figura 15.11); interfața web este accesată, în mod implicit, prin intermediul URL-ului http://localhost:631/.

Figura 15.9: Gestiunea imprimantei în GNOME

Configurarea imprimantei poate fi realizată numai de un utilizator privilegiat. Operațiile uzuale sunt adăugarea imprimantei, ștergerea imprimantei, modificarea configurării de imprimantă. CUPS are instalate majoritatea driverelor de imprimantă astfel încât majoritatea imprimantelor vor putea fi configurate fără operații suplimentare. În caz de probleme sau în cazul folosirii unor imprimante ale căror drivere nu sunt prezente se recomandă accesarea site-ului OpenPrinting[6].

[1] http://libtorrent.rakshasa.no/
[2] http://www.gnu.org/software/ncurses/ncurses.html
[3] http://http://wiki.limewire.org/
[4] http://en.wikipedia.org/wiki/Gnutella
[5] http://en.wikipedia.org/wiki/Portable_Document_Format
[6] http://www.linuxfoundation.org/collaborate/workgroups/openprinting

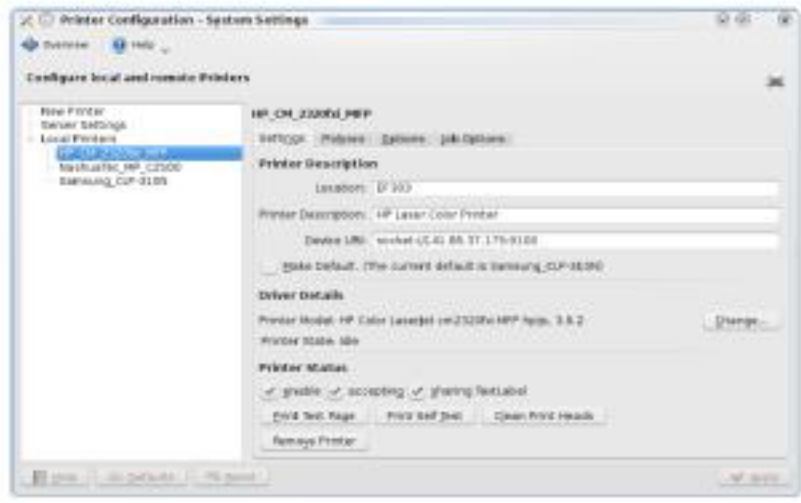


Figura 15.10: Gestiunea imprimantei în KDE

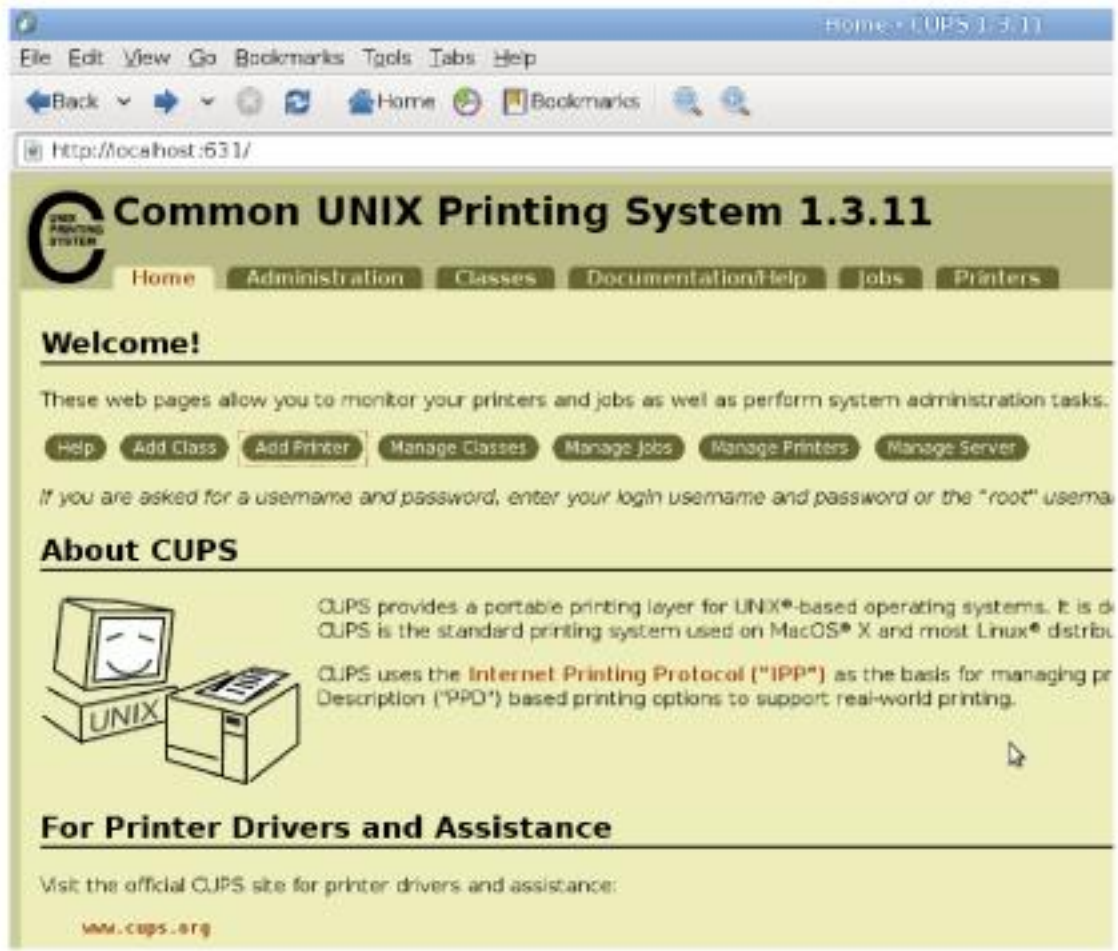


Figura 15.11: Interfața web pentru CUPS

15.7 Jocuri în Linux

Datorită originilor și mediului căruia i-a fost destinat până nu demult, Linuxul nu a reprezentat un segment de piață atractiv pentru producătorii de jocuri. De aceea nu există foarte multe jocuri care pot rula pe acest sistem de operare.

În Windows, cele mai multe jocuri folosesc pentru desenarea obiectelor 3D API-ul Direct3D de la Microsoft, parte a API-ului multimedia DirectX (DirectX a ajuns la versiunea 10 odată cu apariția Windows Vista). Pe lângă acest API (proprietar Microsoft), există și jocuri care folosesc un alt API, open source, numit OpenGL (Open Graphics Library). OpenGL este o colecție de funcții gândite a fi cross-language și cross-platform. Este printre puținele API-uri standardizate folosite în Linux pentru crearea obiectelor 3D și implicit a jocurilor 3D.

Dintre cele mai cunoscute jocuri pe Linux avem:

- Battle for Wesnoth[1] – un joc de strategie fantasy
- Nexuiz[2] – shooter, clonă Quake
- Armagetron Advanced[3] – un joc bazat pe un film SF din anii 80

Figura 15.12: Nexuiz

În afară de aceste jocuri gratis, făcute de comunități conectrate în jurul lor, există și jocuri cu suport pe Linux din partea unor firme mari. Dintre acestea probabil cea mai bine cunoscută este Id Software care oferă majoriatatea jocurilor produse și pe Linux. Epic games, productătorul seriei Unreal, oferă de asemenea pentru anumite jocuri și o variantă pentru Linux.

Wine[4] este o aplicație ce permite rularea unor programe destinate platformelor Windows pe un sistem Linux. Pentru a vedea dacă o aplicație rulează în wine și alte informații legate de rularea ei în wine puteți consulta baza de date a proiectului[5]. Winetricks[6] este un script făcut pentru instala mai ușor anumite aplicații specifice Windows, cum ar fi directx sau framework-ul .NET.

15.8 Dual-monitor în Linux

Odată cu trecerea de la monitoarele CRT la cele LCD și în același timp cu scăderea prețului la hardware, posibilitatea de a conecta două monitoare la sistemul fizic a devenit accesibilă. Creșterea productivității și a confortului în utilizarea calculatorului îndeamnă din ce în ce mai mulți utilizatori să folosească o astfel de soluție.

[1] http://www.wesnoth.org
[2] http://www.alientrap.org/nexuiz
[3] http://armagetronad.net
[4] http://www.winehq.org/
[5] http://appdb.winehq.org/
[6] http://wiki.winehq.org/winetricks

Atunci când sunt conectate două monitoare la sistem (sau un monitor și un videoproiector) acestea pot fi utilizate în două configurații:

- *mirror*, în care amândouă monitoarele arată aceeași imagine
- *extended*, în care imaginile arătate de cele două monitoare sunt diferite și alăturate

Configurația *mirror* este utilă numai în cazul în care se dorește vizualizarea aceleiași informații pe mai multe monitoare. În schimb, configurația extinsă permite mărirea spațiului de lucru, așa cum se poate vedea în figura 15.13.

Figura 15.13: Alăturarea a două desktop-uri. Fiecare este redat pe un monitor separat

Pentru realizarea unei configurații de acest gen se pot folosi utilitare grafice sau utilitare în linie de comandă. În cazul GNOME, accesând `System -> Preferences -> Display` se pot configura atât parametrii pentru fiecare din monitoare (rezoluție, rată de *refresh*) cât și activarea sau dezactivarea fiecărui monitor (figura 15.14). În plus, intrefața de configurare permite alegere modului în care sunt așezate cele două desktop-uri: unul lângă altul, unul deasupra altuia sau, spre exemplu, pe ce monitor se afișează desktop-ul din stânga.

În cazul în care se dorește realizarea unui set de configurări avansate, va trebui folosit utilitarul **`xrandr`**, descris în secțiunea 13.4.1.

Un exemplu de comandă xrandr ce permite configurarea monitorului extern pentru un laptop ca desktop extins este următoarea:

```
1 george@asgard:~$ xrandr --output VGA --mode 1920x1200 --rate 60 --pos 0x0
   --output LVDS --mode 1280x800 --rate 60 --pos 0x1200
```

Așa cum se poate vedea, pentru fiecare din monitoare (`LVDS` – ecranul laptop-ului și `VGA` – ieșirea video) se configurează trei parametri:

- rezoluția, prin parametrul `--mode`
- rata de refresh, prin parametrul `--rate` (acest parametru poate lipsi, caz în care sistemul de operare va alege automat cea mai bună variantă posibilă)
- poziția desktop-ului, prin parametrul `--pos`

Poziția desktop-ului se măsoară, la fel ca majoritatea elementelor ce țin de grafica pe calculator, începând din colțul din stânga sus, având axa `Ox` orizontală și axa `Oy`

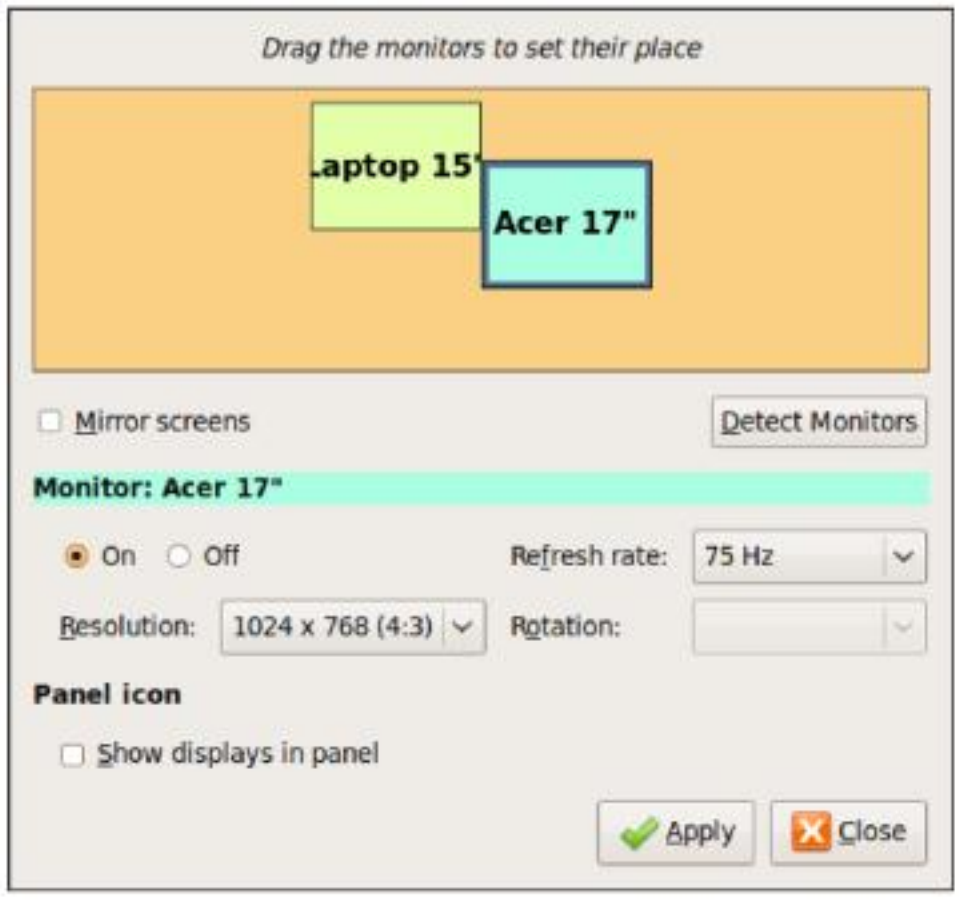


Figura 15.14: Configurări pentru desktop extins in GNOME

orientată în jos. Astfel, în exemplul de mai sus, ieșirea `VGA` se va afla cu colțul din stânga sus în origine (`X=0`, `Y=0`), iar ieșirea `LVDS` se află sub cea `VGA`, amândouă aliniate la stânga (`X=0`, `Y=1200`).

Un alt exemplu de configurare a unui desktop multiplu, care a generat configurația prezentă în figura 15.14, este:

```
1 george@asgard:~$ xrandr --output VGA --mode 1280x1024 --pos 1280x0 --
  output LVDS --mode 1280x800 --pos 0x224
```

Ieșirea `LVDS` este plasată în stânga (`X=0`), cu 224 de pixeli mai jos ca originea (`Y=224`). Ieșirea `VGA` este plasată în dreapta, la 1280 de pixeli departare de origine (`X=1280`) și la același nivel cu aceasta (`Y=0`). Cei 224 de pixeli, calculați ca diferența dintre înălțimea celor două ecrane (1024 și 800), au fost folosiți pentru a aduce partea de jos a celor două ecrane la același nivel. După cum se poate vedea și în figura 15.14, în patea de sus a ecanului mai mic (`LVDS`) apare o bandă neagră care nu va fi observată însă pe niciunul din monitoare.

15.9 Documente office în Linux

Deși suita Microsoft Office nu este disponibilă pentru Linux, interoperabilitate aproape completă cu ea se poate obține utilizând OpenOffice.

OpenOffice are, similar cu produsul Microsoft, componente pentru editarea documentelor, a foilor de calcul, gestiunea bazelor de date și realizarea prezentărilor, numite Writer, Calc, Base, respectiv Impress. În plus, există un corespondent al lui Visio numit Draw și un editor de formule matematice numit Math.

Versiunea 3.0 a OpenOffice are inclusiv suport pentru documente în format OOXML[1]. Este recomandată, în schimb, în situația în care nu este nevoie de compatibilitate cu Microsoft Office, folosirea documentelor în format nativ. Standardul se numește ODF (*Open Document Format*), este omologat ISO, și implementat de numeroase alte suite similare.

Pentru distribuția documentelor *read-only* este recomandat formatul `.pdf`. Salvarea unui document din Writer în acest format este o facilitate care vine cu OpenOffice, nefiind nevoie de instalarea unui plugin.

Desigur, există alternative la OpenOffice; KOffice este o suită similară, dar sunt și multe programe *standalone*, care pot edita un singur tip de fișier. Asemenea programe se disting prin viteză de operare superioară și necesități hardware mai reduse. Două exemple cunoscute sunt AbiWord[2] și Gnumeric[3].

Cuvinte cheie

- Rhythmbox
- Amarok
- Banshee
- codecuri
- VLC
- mplayer
- wodim (cdrecord)
- genisoimage
- GnomeBaker
- K3b
- Pidgin
- BitTorrent
- Transmission
- Ktorrent
- CUPS
- OpenOffice
- ODF

Întrebări

1. Care dintre următoarele afirmații este adevărată despre codec-urile audio și video în Linux?
   - ❑ nu există codec-uri audio/video pentru Linux
   - ❑ un utilizator trebuie să plătească pentru codec-uri în Linux
   - ❑ codec-urile sunt gratuite, dar trebuiesc compilate din surse
   - ❑ majoritatea codec-urilor sunt instalate ca dependențe ale player-elor corespunzătoare
2. Care dintre următoarele comenzi NU are legătură cu procesul de scriere a unui CD sau DVD?

---

[1] Formatul OOXML a fost introdus odată cu Microsoft Office 2007 și se distinge prin adăugarea literei "x" la extensia veche, de exemplu `.docx`

[2] http://www.abisource.com/

[3] http://projects.gnome.org/gnumeric/

- ❑ **wodim**
- ❑ **cdrecord**
- ❑ **mplayer**
- ❑ **genisoimage**

3. Ce program are și facilități pentru IRC?
   - ❑ Pidgin
   - ❑ Ktorrent
   - ❑ centerim
   - ❑ Audacious
4. CUPS este o suită de programe care se folosește la:
   - ❑ printare
   - ❑ scrierea unui CD/DVD
   - ❑ editarea de documente Office
   - ❑ CUPS este un joc
5. Pentru trimiterea unui document către un coleg pentru printare este recomandat formatul:
   - ❑ `.doc`
   - ❑ `.docx`
   - ❑ `.pdf`
   - ❑ `.odt`

Anexa A

Răspunsuri la întrebări

Capitolul 1. Introducere

1. `interpretor de comenzi` – trebuie să se poată interpreta comenzi pentru funcționarea sistemului de operare
2. `BIOS` – BIOS-ul nu este una din resursele de care este nevoie în virtualizare
3. `Larry Wall` – autorul limbajului Perl
4. `căutarea si eliminarea programelor virus` - de aceste lucruri se ocupă o aplicație antivirus
5. `OpenSolaris` – celelalte sunt sisteme de operare specializate pentru dispozitive mobile
6. `OpenVZ` – rulează doar sub Linux
7. `un proiect care produce o componentă importantă a aplicațiilor ce rulează peste nucleul Linux` - GNU's not Unix
8. `Windows` – pentru că viața e dură și marketing-ul e marketing
9. `Symbian OS` – dar iPhone OS is catching up
10. `Unix` – dezvoltat de un grup de la AT&T în anul 1969

Capitolul 2. Instalarea Linux. Configurări de bază

1. `1` – poate exista doar un root
2. `configurarea unei partiții primare` – pentru că se rulează într-un mediu temporar
3. `PalmOS`
4. `adevărat, adevărat`
5. `comanda a fost lansată de utilizatorul root` – se presupune că root are privilegii complete
6. `swap` – swap este un tip de partiție, nu un tip de sistem de fișiere

7. `virtuală` – toate celelalte sunt tipuri de partiții
8. `Ubuntu`
9. `window manager` – window manager se referă la mediul grafic
10. `adduser` – adduser este folosită pentru a adăuga utilizatori

Capitolul 3. Gestiunea pachetelor și utilizatorilor

1. `useradd` – va crea un utilizator nou folosind configurația implicită
2. `Permisiunile asupra /etc/passwd nu asigură securitate maximă.` – `/etc/passwd` poate fi citit de orice utilizator
3. `apt este un wrapper peste dpkg`
4. `legături simbolice`
5. `grupurile din care face parte` – informațiile despre grupuri sunt ținute în fișierul `/etc/group`
6. `conţine versiuni ale unor pachete diferite de ceea se găsește în repository-urile centrale` – programatorii folosesc astfel de repository-uri pentru testarea de noi versiuni ale programelor

Capitolul 4. Sisteme de fișiere

1. `suportului pentru jurnalizare` – în cazul în care se întrerupe curentul, ultimele operații pot fi refăcute pe baza jurnalului
2. `/etc/fstab`
3. `606 - 653 = rw-r-x-wx`; `a-x` va elimina dreptul de execuție de la toți (`rw-r---w-`); `u+x` va da drept de scriere owner-ului (nu se realizează schimbări); `g-r` va elimina drepturile de citire ale grupului (`rw----w-`); `o+r` va da drepturi de citire altor utilizatori; drepturile în final vor fi `rw----w-` (606)
4. `cat` – comanda **`cat`** poate fi aplicată doar fișierelor
5. `interfață de rețea de looback` – nu are corespondent în sistemul de fișiere
6. `EXT3`
7. `toți utilizatorii` – `755 = rwx-r-xr-x`
8. `afișarea de mesaje despre arhivare` – conform **`man`**
9. `mount`
10. `formatarea partiției cu sistemul de fișiere dorit` – fiind dată o partiție, pentru a putea folosi spațiul de pe aceasta, este necesară formatarea acesteia cu un sistem de fișiere

Capitolul 5. Procese

1. `init` – este primul proces pornit în sistem, cu `PID`-ul 1
2. `nohup` – comanda este folosită pentru a rula un proces cu caracteristici de daemon
3. `1` – există un singur proces `init` cu `PID`-ul 1
4. `oricâte` – nu există limită pentru procesele **`bash`** din sistem
5. `kill` – comanda doar trimite semnale proceselor
6. `fals, fals` – un program poate fi rulat de mai multe ori – astfel se vor crea mai multe procese care au aceeași imagine; **`top`** afișează pe prima coloană pid-ul
7. `pkill` – comanda permite trimiterea de semnale
8. `/proc` – un sistem de fișiere virtual, în care se găsesc informații despre fiecare proces în parte
9. `adevărat, fals` – toate PID-urile au valori pozitive; în mod normal, prioritățile au valori între -19 și +20
10. `explorer - nice` – Explorer este un shell, **`nice`** este o comandă pentru modificarea priorității unui proces

Capitolul 6. Pornirea și inițializarea sistemului

1. `BIOS, GRUB, vmlinuz, init` –
2. `un sector ce conține semnătura 0xAA55 (ultimii doi octeți)` – un sector este de tipul boot dacă el conține acești ultimi doi octeți, astfel se poate face un sector care să nu conțină un sistem de operare dar să fie bootabil având acești ultimi doi octeți
3. `upstart` –
4. `4, 1, 1, oricâte` –
5. `POST/CMOS` –
6. `stage2` –
7. `/boot/grub/menu.lst` –
8. `init, getty, login, bash` –
9. `(hd1,2)` –
10. `adevărat/adevărat` –

Capitolul 7. Analiza hardware a sistemului

1. `CPU` – CPU (Central Processing Unit) este procesorul, echivalentul unității de comandă și a unității de execuție din modelul Von Neumann

2. `dd` – **dd** copiază date *raw* de pe un dispozitiv
3. `dd` – **dd** este utilitar de copiere de date, nu de informare
4. `modinstall` – nu există comanda **`modinstall`**
5. `/dev/hda3` – /dev/hda3 reprezintă un disc
6. `/dev/ttyS0` – /dev/ttyS0 este asociat unui dispozitiv serial
7. `southbridge` – southbridge este unul din cele două chipset-uri importante de pe placa de bază
8. `TGZ` – TGZ se referă la un tip de arhivă
9. `placa grafică` – AGP (Advanced **Graphics** Port)
10. `dimensiunea registrelor` – dimensiunea registrelor este cea care denotă tipul arhitecturii (arhitectură pe 32 de biți, pe 64 de biți)

Capitolul 8. Configurări de rețea

1. `Serverul web de la adresa cs.pub.ro este oprit` – stația este online (comanda ping reușește); dacă browser-ul nu reușește înseamnă că serverul web este oprit
2. `Nu` – adresa este greșită (al treilea octet este 257; mai mare decât 255)
3. `172.16.150.0` – se face operația 172.16.150.200 ȘI-LOGIC 255.255.255.0; rezultă 172.16.150.0
4. `Dorel a generat foarte mult trafic în timp scurt, lucru specific virușilor` – opțiunea -f a comenzii ping generează foarte multe pachete (flood)
5. `Asocia nume de domenii cu adrese IP` – conform definiției
6. `102.168.124.159, pentru că a fost ultima adresă configurată de root` – masca este configurată implicit dacă nu se folosește opțiunea netmask
7. `fals, adevărat` – fișierul se cheamă /etc/network/interfaces
8. `adevărat, false` – serverele pot fi accesate; DNS este necesar pentru a asigura referirea acestora prin nume (în loc de adresă IP)
9. `calea către sursă și destinație` – conform definiției
10. `ping localhost` – toate celelalte comenzi folosesc comanda ping catre un server referit prin nume; referirea prin nume necesita un server DNS

Capitolul 9. Servicii de rețea

1. `editarea unui nou mesaj` – editarea se face pe sistemul local; nu necesită o conexiune client-server

2. `fals, fals` – telnet este protocol de nivel aplicație care funcționează peste TCP (nivel transport); comunicația este bidirecțională
3. `SSH` – SMTP și IMAP sunt protocoale de e-mail, HTTP este protocol web
4. `FTP` – FTP (File Transport Protocol) nu are nici o legătură cu serviciul web
5. `Postfix` – Firefox este browser, Outlook client de e-mail, Apache este server web
6. `pentru stabilirea conexiunii utilizatorul bestman trebuie să existe pe test.com` – comunicația FTP necesită autentificare pe server; bestman este utilizatorul care trebuie să se autentifice
7. `adevărat, fals` – din definiție
8. `Thunderbird` – Thunderbird este client de e-mail
9. `ssh` – wget și curl sunt clienți web, iar telnet permite conectarea și transmiterea de comenzi și răspunsuri HTTP
10. `niciuna dintre variante` – comunicația se realizează folosind SMTP

Capitolul 10. Elemente de securitate

1. `filtrarea pachetelor de rețea`
2. `citire și execuție` – cu **`umask 022`** fișierul va fi creat cu drepturi `644`. După execuția comenzii **`chmod go+x test`** grupul va primi drept de execuție. Într-un final, grupul va avea drepturile `r-x`
3. `_cOrN31[]sh+` – este o parolă complexă, formată cu cifre, litere și simboluri
4. `vârfuri de tensiune în rețeaua electrică` –
5. `toate variantele`
6. `fals, fals` – virusul dintr-un e-mail trebuie întâi activat; în funcție de tipul lui și de clientul de e-mail, metoda de activare poate diferi; un sistem poate fi "gazda" oricâtor viruși, atât timp cât nu este protejat
7. `numărul de clienți ai server-ului` – restul informațiilor se pot extrage direct din traficul monitorizat
8. `ulimit`
9. `definirea unei politici de securitate`
10. `/etc/shadow`

Capitolul 11. Compilare și linking

1. `ajută ca aplicațiile să fie mai simplu de întreținut` – programatorii nu mai trebuie să dezvolte și să întrețină componente, utilizând componentele disponibile în biblioteci

2. `-c`
3. `se numește portabilitate`
4. `make <target>` – fișierul `Makefile` trebuie să se găsească în directorul curent; pentru trimiterea unui fișier `Makefile` ca parametru se folosește parametrul `-f`
5. `fișiere sursă C fără directive de precompilare`
6. `static library` – codul dintr-o biblioteca statică este integrat la compilare în executabil
7. `adevărat, fals` – apelurile de sistem pot fi executate și direct de către aplicații
8. `fișierul este executabil` – restul informațiilor reies din ieșirea comenzii **`file`**
9. `fals, fals` – programele interpretate se depanează mai ușor decât cele compilate deoarece procesul de dezvoltare nu mai presupune compilarea codului; în urma compilării nu rezultă fișiere în limbaj de asamblare (fișierele în limbaj de asamblare nu se pot executa direct pe procesor)
10. `strace <program>`

Capitolul 12. Shell scripting

1. `Flash` – *Flash* este un plug-in pentru web browser
2. `grep` – este singura comandă dintre cele prezentate care poate primi ca parametru o expresie regulată
3. `<<` – este operatorul pentru deschiderea unui *here document*
4. `test a ==b` – `[$a == $b]` nu are operanzii separați de paranteze; celelalte opțiuni nu sunt folosite pentru testarea egalității
5. `Sortează liniile în format numeric`
6. `cut` – **`cut`** este folosit pentru extragerea de componente dintr-o linie de text
7. `Viteza mare de execuție` – scripturile shell trebuie să fie interpretate, motiv pentru care au o viteză mică de execuție
8. `Prezintă calea către utilitarul folosit pentru rularea scriptului`
9. `repeat` – nu există comanda `repeat` în Bash
10. `find . -mindepth 2 -type f` – comanda afișează recursiv doar fișierele din subdirectoare

Capitolul 13. Mediul grafic

1. `PNG` – PNG este un format bitmap cu suport pentru transparență (*alpha channel*)

2. `UTF-8` – `UTF-8` este un standard de codificare a caracterelor în fișiere
3. `permite unui utilizator să se autentifice în interfața grafcă` – conform definiției
4. `Xorg și bitmap` – Xorg este o implementare de server X, bitmap este un tip de imagini
5. `Dorel nu are dreptul să pornească serviciul kdm` – din prompt se poate observa că utilizatorul nu este autentificat ca `root`
6. `Spotlight` – Spotlight este un serviciu de indexare și căutare a fișierelor în Mac OS X
7. `KDE` – KDE este un *Desktop Environment*
8. `System Information` – restul aplicațiilor sunt folosite pentru a afla informații în linia de comandă
9. `Adevărat, Fals` – Un Window Manager controlează cum arată ferestrele aplicațiilor prin intermediul componentei *window decorator*. Clientul X nu interacționează cu utilizatorul direct; server-ul X face acest lucru.
10. `Xorg` – Xorg este o implementare de server X

Capitolul 14. Utilitare pentru dezvoltare

1. `Vim` – **`gdb`** este un debugger, **`ctags`** este un program utilizat pentru a prelucra o sursă și scoate un fișier cu indecși și **`astyle`** este un *code beautifier*
2. `Valgrind` – **`git`** este un program de *revision control*, **`indent`** este un *code beautifier* și Trac este un utilitar de management a proiectelor
3. `boot` – `checkout/clone` aduc local o copie a repository-ului, `commit` adăugă modificările la repository și `update` aduce repository-ul local la cea mai nouă versiune
4. `licențierea codului sub GPL` – o licență nu va influența frumusețea sau lizibilitatea codului.
5. `OpenOffice` – OpenOffice este Office Suite, nu oferă unelte necesare dezvoltării codului.
6. `ctags` – **`splint`** este un utilitar de analiză statică a codului, **`indent`** este un code *beautifier* și **`gdb`** este un utilitar de depanare
7. `graphic` – în modul `visual` se intră apăsând tasta *v*, în modul `command` se intră apăsând :, în modul `insert` se intră apăsând tasta *i*
8. `:w` – *:q* închide editorul, *:s* va face replacei, *:help* lansează într-un tab fișierul de help
9. `git` – Vim este un editor, Eclipse este un IDE și ddd este un debugger
10. `Trac` – GTK este un *widget toolkit*, ddd este un *debugger* și Waterfall este un model de a dezvoltare software-ului

Capitolul 15. Viața în Linux

1. `Majoritatea codec-urilor sunt instalate ca dependenţe ale player-elor corespunzătoare.` – pentru o utilizare mai ușoară (nu trebuie instalare ulterior codecurile)
2. `mplayer` – este un media player
3. `Pidgin` – `libpurple`, biblioteca utilizată de Pidgin are suport și pentru `IRC`
4. `printare` – *Common Unix Printing System*
5. `.pdf` – celelalte formate nu sunt suportate integral pe toate sistemele

Bibliografie

[1] Tom Adelstein and Bill Lubanovic. *Linux System Administration*. O'Reilly Media, March 2007.

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *compilers: Principles, Techniques, and Tools*. Addison Wesley, January 1986.

[3] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, second edition, April 2008.

[4] Nitesh Dhanjani and Justin Clarke. *Network Security Tools*. O'Reilly Media, April 2005.

[5] Bruce Eckel. *Thinking in Java*. Prentice Hall PTR, fourth edition, February 2006.

[6] John Fusco. *The Linux Programmer's Toolbox*. Prentice Hall, March 2007.

[7] Simson Garfinkel, Gene Spafford, and Alan Schwartz. *Practical Unix and Internet Security*. O'Reilly Media, third edition, February 2003.

[8] Mark G. Graff and Kenneth R. Van Wyk. *Secure Coding: Principles and Practices*. O'Reilly, July 2003.

[9] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, fourth edition, September 2006.

[10] Michael Horton and Clinton Mugge. *Hacknotes - Network Security Portable Reference*. McGraw-Hill Osborne Media, July 2003.

[11] Steve Hunger. *Debian GNU/Linux Bible*. Wiley, May 2001.

[12] Brian W. Kernighan and Rob Pike. *The Practice of Programming*. Addison-Wesley Professional, February 1999.

[13] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, second edition, April 1988.

[14] Donald E. Knuth. *Art of Computer Programming*. Addison-Wesley Professional, October 1998.

[15] John R. Levine. *Linkers and Loaders*. Morgan Kaufmann, October 1999.

[16] Robert Love. *Linux Kernel Development*. Novell Press, second edition, January 2005.

[17] Robert Love. *Linux System Programming*. O'Reilly Media, September 2007.

[18] Neil Matthew and Richard Stones. *Beginning Linux Programming*. Wrox, November 2007.

[19] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, fourth edition, November 2008.

[20] Cyrus Peikari and Anton Chuvakin. *Security Warrior*. O'Reilly Media, January 2004.

[21] Eric S. Raymond. *The Art of Unix Programming*. Addison-Wesley Professional, October 2003.

[22] Răzvan Rughiniș, Răzvan Deaconescu, Andrei Ciorba, and Bogdan Doinea. *Rețele locale*. Printech, September 2008.

[23] Răzvan Rughiniș, Răzvan Deaconescu, George Milescu, and Mircea Bardac. *Utilizarea sistemelor de operare*. Printech, September 2007.

[24] Joel Scambray, Mike Shema, and Caleb Sima. *Web Applications (Hacking Exposed)*. McGraw-Hill Osborne Media, second edition, June 2006.

[25] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, second edition, October 1996.

[26] Avi Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. John Wiley & Sons, seventh edition, December 2004.

[27] W. Richard Stevens. *Advanced Programming in the Unix Environment*. Addison-Wesley Professional, June 1992.

[28] W. Richard Stevens. *TCP/IP Illustrated*. Addison-Wesley Professional, January 1994.

[29] W. Richard Stevens. *Unix Network Programming*. Prentice Hall, second edition, January 1998.

[30] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, fourth edition, August 2002.

[31] Andrew S. Tanenbaum. *Structured Computer Organization*. Prentice Hall, fifth edition, June 2005.

[32] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, third edition, December 2007.

[33] Larry Wall, Tom Christiansen, and Jon Orwant. *Programming Perl*. O'Reilly Media, third edition, July 2000.

Glosar

-Wall, 326
-- help, 43
--help, 372
.vimrc, 477
/dev, 201
/etc/hostname, 236
/etc/hosts, 235
/etc/network/interfaces, 225
/etc/resolv.conf, 233
/proc/cpuinfo, 208
/proc/modules, 194
înlănțuirea comenzilor, 382
întrerupere, 204
__cplusplus, 361

ACL (*Access Control List*), 88
ACPI (*Advanced Configuration and Power Interface*), 198
adresă IP, 219
adresă IPv6, 219
adresă MAC, 219
agile, 497
Anjuta, 469, 494
apel de sistem, 186
apeluri de sistem, 487
API (*Application Programming Interface*), 7
aplicații de bază, 2
apropos, 43, 372
arhitectura unui sistem de calcul, 188
arhivare, 82
asamblare, 328, 333
astyle, 472
autocompletion (bash), 369
AVFS (*Anti-Virus Filesystem*), 103
awk, 410

background, 130
bash, 366
bg, 132
bibliotecă, 341
bibliotecă partajată, 342, 486
bibliotecă statică, 342
biblioteci cu legare dinamică, 342
biblioteci cu legare statică, 342
BIOS (*Basic Input Output System*), 153
bitmap, 442
booting, 152
bootloader, 154
branch, 479
bug, 489, 497
bug-report, 497

C și C++, 357
Căsuță poștală, 264
cache L1, 189
cache L2, 189
cachegrind, 492
cale absolută, 72
cale relativă, 72
case (bash), 367, 394
cat, 378, 400
certificat digital, 311
chainloading, 159
checkout, 479
chmod, 291
chown, 291
chroot, 300
CLI (*Command Line Interface*), 38, 366
client, 254
cmdlets, 434
CMOS (*Complementary Metal Oxide Semiconductor*), 153
code folding, 473
coding style, 470
comandă, 42
commit, 479
compilare, 321, 328, 331
compilare din surse multiple, 327
compilator, 321, 324

configurare bash, 426
configurare prompt shell, 429
configurare vim, 477
configure, 484
consolă, 40
consola GRUB, 164
controlul accesului, 281
controlul versiunii, 479
criptare, 310
csh, 366
ctags, 478, 482
CTRL-\, 139
CTRL-ALT-DEL, 142
CTRL-C, 139
CTRL-Z, 131, 139
cut, 403
Cygwin, 356

daemon, 133
darcs, 479
dd, 206
DDD (*Data Display Debugger*), 489, 491
Debian, 25
debug, 489
debugger, 489
depanare, 489
depmod, 195
desktop environment, 447
dezasamblare, 334
dhclient, 225
DHCP (*Dynamic Host Control Protocol*), 251
director, 69, 74
display manager, 447
dispozitiv bloc, 203
dispozitiv caracter, 203
dispozitive de intrare/ieșire, 187
distribuție Linux, 24
DLL (*Dynamic-link library*), 342
DMA, 206
DNS (*Domain Name System*), 223, 251
documentație html, 499
documentație LaTeX, 499
DoS (*Denial of Service*), 281
Doxygen, 499
driver, 3, 193

echo, 377
Eclipse, 469, 494
ELF (*Executable and Linking Format*), 340
Emacs, 320
emulare, 17
emulator de terminal, 41
EncFS, 103
Ethernet, 252
ethtool, 239
exit, 377
export, 377
expresie regulată, 407
extern, 361
Extreme Programming, 497

feature creep, 282
fg, 132
fișier, 69, 74
fișier executabil, 339
fișier obiect, 333
find, 416
Firewall, 310
firewall, 310
for (bash), 367, 395
foreground, 130
fork bomb, 281
frecvență, 188
fsck, 212
FTP (*File Transfer Protocol*), 251
funcții bash, 424
FUSE, 103

gateway, 222
GCC (*GNU Compiler Collection*), 325
gcc în Windows, 356
GDB (*GNU Debugger*), 489
GID (Group Identifier), 55
git, 479, 481
GNU, 23
GNU Autotools, 485
GNU CPP (*GNU C preprocessor*), 331
GNU/Linux, 23
GParted, 49
grep, 405, 407
GRUB, 162
GUI, 7
GUI (*Graphical User Interface*), 39
GUI (grafical user interface), 366

hard-disk, 190
hardware, 186

head, 402
helgrind, 492
here document, 380
here string, 382
hipervizor, 16
host, 234
hostname, 236
HTML (*Hypertext Markup Language*), 270
HTTP (*Hypertext Transfer Protocol*), 251, 270
hwinfo, 198

ICMP, 236
IDE (*Integrated Development Environment*), 320, 469, 494
IDS (*Intrusion Detection System*), 312
if (bash), 367, 391
ifconfig, 225
imagine de CD, 212
imagine vectorială, 443
IMAP (*Internet Mail Access Protocol*), 251, 264
indent, 471
info, 43, 372
init, 117, 143
insmod, 195
instalarea din surse, 484
interfață de rețea, 225
interpretare, 321
interpretor, 321
IP, 218
ipconfig, 240
IRC (*Internet Relay Chat*), 501
istoric (bash), 369

Javadoc, 499
jeton hardware, 288
jobs, 132
jurnalizare, 93

kernel, 2, 185
kill, 137
killall, 138
kswapd, 143

LAN (*Local Area Network*), 218
ld, 486
ldconfig, 486
ldd, 486
legătură simbolică, 74
libnotify, 460
limbaj asamblare, 332
limbaje de nivel înalt, 322
limbaje de nivel scăzut, 322
limbaje de programare de nivel înalt, 322
limbaje de programare de nivel scăzut, 322
link-editare, 328, 337
linker, 338
Linux, 23
Live CD, 31
locate, 415
loopback, 226
lsmod, 194
lspci, 197
lsusb, 196

mașină virtuală, 16
make, 345
man, 43, 372
MAN (*Metropolitan Area Network*), 218
management proiecte software, 496
massif, 492
Master Boot Record, 31
MBR (*Master Boot Record*), 156, 211
medii integrate de dezvoltare, 494
memcheck, 492
memorie, 187
memorie cache, 189
memorie virtuală, 285
MinGW (*Minimalist GNU for Windows*), 356
modprobe, 195
module de kernel, 194
montarea unui sistem de fișiere, 97
MSVC (*Microsoft Visual C++*), 469, 502
multi-core, 188
multiprogramare, 116
multitasking, 116

name-mangling, 360
nice, 146
nl, 400
nm, 335
nohup, 134
northbridge, 191
NTFS-3G, 103
NTLoader, 178

objdump, 334

one liner, 378
operatorul &, 130
operatorul $((...)), 420
operatorul >>, 380
operatorul <<<, 382
operatorul <<, 380
operatorul |, 140, 382
optimizare, 335
optimizarea compilării, 335

pachet software, 58
pachete necesare pentru dezvoltarea codului C, 324
pager, 77
paginator, 77
parametrii unui script shell, 421
partiți logică, 31
partiție, 30
partiție de swap, 30
partiție extinsă, 31
partiție primară, 31
PATH, 74
pdflush, 143
peer to peer, 254
pgrep, 123
PID, 113
ping, 236
PKI (*Public Key Infrastructure*), 287
pkill, 138
placa de bază, 191
Poșta electronică, 263
poff, 245
pon, 245
POP3 (*Post Office Protocol*), 251, 264
port, 255
port I/O, 205
portabilitate, 354
POSIX (*Portable Operating System Interface*), 356
POST (*Power-on Self Test*), 154
PPP, 243
PPPoE, 243
pppoeconf, 243
preprocesare, 328, 330
principiul celui mai mic privilegiu, 281
principiul limitării drepturilor, 281
printf, 377
proces, 112
proces de dezvoltare software, 496
procesor, 188
procfs, 114, 127
program, 112
prompt, 42
protocol, 249
ps, 117

RAM, 190
ramdisk, 32
raster, 442
rețea de calculatoare, 217
read, 378
readline, 368
Redmine, 497
registre, 188, 189
registre vim, 476
repository, 479, 497
resolvconf, 233
reverse search (bash), 369
rmmod, 195
roadmap, 497
root, 45
ruter, 219, 252

S.M.A.R.T., 200
scanarea porturilor, 307
scheduling, 115
schimbare de context, 116
sector de boot, 154
sed, 408
Semantic Web, 268
semnal, 135
semnale, 487
seq, 396
server, 254
set, 377
setgid, 299
setuid, 299
sfdisk, 211
shebang, 375
shell, 41, 365
shell script, 365
shift, 422
shutdown, 44
SIGKILL, 135
SIGSEGV, 135
SIGTERM, 135
simbolul $, 419
sistem de fișiere, 70
sistem de operare, 1
sistem multi-core, 115

sisteme centralizate de versionare, 479
sisteme distribuite de versionare, 479
sisteme multi-tasking, 5
sisteme multi-user, 5
smartmontools, 200
SMTP (*Simple Mail Transfer Protocol*), 251, 265
SNMP (*Simple Network Management Protocol*), 251
SO (sistem de operare), 1
southbridge, 191
spațiu kernel, 186
spațiu utilizator, 186
spam, 267
splint, 482
sshfs, 103
starea unui proces, 115
stivă de protocoale, 249
strace, 488
su, 46
Subversion, 479, 480
sudo, 47, 301
SVN (Subversion), 479, 480
swap, 30, 213
swapping, 30, 141
switch, 252
syntax highlighting, 473
sysctl, 210
sysfs, 196
system tray, 460

tabela de partiții, 157
tac, 400
tag, 479
tail, 402
Task manager, 142
TCP (*Transmission Control Protocol*), 251
terminal virtual, 40
test (bash), 391
TFTP, 254
time, 126
top, 124
tr, 404
Trac, 497
traceroute, 237
troian, 306
trunchiere fișier, 382
TTL, 237
TUI (*Text User Interface*), 7, 39

Ubuntu, 25
udev, 201
UDP(*User Datagram Protocol*), 251
UID (User Identifier), 55
ulimit, 302
umask, 292
Unicode, 444
unitatea aritmetică și logică, 187
unitatea de comandă, 187
update, 479
Usenet, 501
utilizare biblioteci, 344

valgrind, 492
variabila `$?`, 392
variabila `$@`, 422
variabila `$#`, 421, 422
variabile make, 352
variabile speciale shell, 423
VBR (*Volume Boot Record*), 156
VCS (Version Control System), 479
vectorial, 443
vi, 320
vim, 320, 473
vimtutor, 474
virtualizare, 16
virus, 305
VNC (Virtual Network Computing), 462

WAN (*Wide Area Network*), 218
waterfall, 497
wc, 405
webmail, 266
whereis, 372
while (bash), 367, 398
wiki, 497
window decorator, 447
window manager, 446
WLAN, 253
working copy, 479
worm, 306

X11, 444
XHTML, 271

Cuvânt înainte

Noțiunea de sistem de operare reprezintă, probabil, unul dintre termenii cei mai des întâlniți în domeniul calculatoarelor, și nu numai. De la formele greoaie dezvoltate în anii '60, cunoscute doar profesioniștilor, sistemele de operare au cunoscut o transformare continuă, strâns corelată cu dezvoltarea sistemelor de calcul și a tehnologiilor asociate. În ziua de astăzi, sistemele de operare oferă o interfață facilă și prietenoasă atât utilizatorilor obișnuiți ai serviciilor Internet, cât și utilizatorilor comerciali ai aplicațiilor dedicate, celor ce folosesc facilitățile multimedia și jocuri, sau celor profesioniști care dezvoltă aplicații sau întrețin sisteme de calcul și rețele de calculatoare. Evoluția tehnologică a dus la dezvoltarea sistemelor de operare pentru un număr tot mai mare de dispozitive, de la sisteme server, desktop și laptop la PDA-uri și smartphone-uri.

Cartea de față își propune familiarizarea cititorului cu lumea sistemelor de operare și, în particular, cu latura preponderent tehnică a acestora. Am creat această lucrare având în permanență în vedere cunoștințele de bază și cadrul conceptual necesare unui student la o facultate de calculatoare. În această structură, cartea este însă construită pentru a fi utilă oricărui cititor care caută un prim contact cu domeniul sistemelor de operare. Sperăm ca parcurgerea sa să ofere și un set de deprinderi și abordări în soluționarea problemelor care depășesc sfera sistemelor de operare.

Diversitatea subiectelor abordate a reprezentat o dificultate în crearea unei succesiuni clare de capitole. Strategia aleasă este una stratificată, fiecare capitol bazându-se pe cele studiate anterior. Totuși, au existat momente în care a trebuit să utilizăm anumite noțiuni înainte de a fi definit cadrul conceptual, sau la câteva capitole distanță de prezentarea lor. În astfel de situații cititorului îi sunt oferite referințe către capitolele în care sunt clarificate noțiunile invocate.

Cartea urmărește prezentarea și discutarea noțiunilor de bază necesare unui student în primii ani de facultate, în domeniul calculatoarelor. Diversitatea subiectelor și nivelul de detaliu recomandă o asimilare în profunzime a informațiilor, dincolo de durata unui semestru sau a unui an. Sperăm ca studentul dornic de aprofundare să răsfoiască această carte în momentele în care caută sprijin suplimentar pentru rezolvarea unei probleme din domeniu.

Din punct de vedere tehnic, materialul de față oferă o perspectivă ce aparține preponderent universului Linux. Am considerat contactul cu Linux ca pe o oportunitate aparte pentru o majoritate a utilizatorilor ce provin din mediul Windows, în care deseori alternativele în domeniul sistemelor de operare nu reprezintă o opțiune luată în considerare. Dorința noastră este ca utilizarea unui nou sistem de operare, cu o răspândire și o evoluție tot mai intense, să ofere o nouă perspectivă asupra lumii calculatoarelor în general și a sistemelor de operare în particular. Deși cartea este

focalizată pe Linux, fiecare capitol include secțiuni de studii de caz în care sunt prezentate mecanismele similare dintr-un sistem Windows.

Structura cărții este concepută pentru a oferi atât o prezentare a cadrului conceptual, cât și o parte aplicativă construită prin exemple. Fiecare capitol este prefațat de o mică secțiune „Ce se învață în acest capitol?", utilă pentru reperarea principalelor noțiuni. Capitolele se încheie cu o secțiune de „Cuvinte cheie" și apoi de „Întrebări", pentru a permite cititorului o autoevaluare a cunoștințelor dobândite. Unele secțiuni sunt marcate cu simboluri grafice cu o semnificație specială: important, notă, OZN (pentru tehnologiile recente de tipul „bleeding edge") și atom (pentru aspecte tehnice avansate).

Recomandăm parcurgerea secvențială a cărții, dar cititorul avansat poate sări direct la un capitol de interes particular. Dat fiind conținutul practic detaliat, este utilă folosirea calculatorului pentru rularea comenzilor prezentate și pentru explorarea opțiunilor existente, în paralel cu parcurgerea noțiunilor teoretice.

Mulțumim tuturor celor care au contribuit la realizarea cărții. Modul de organizare și prezentare, ca și diversitatea informațiilor prezentate se bazează pe efortul continuu și pasiunea unei echipe entuziaste. În primul rând, îi mulțumim domnului profesor Nicolae Ţăpuș, precum și colegilor noștri Vlad Dogaru, Mihai Maruseac, Daniel Rosner și Andrei Buhaiu, a căror implicare a constituit un beneficiu direct în elaborarea acestei cărți. Mulțumim, de asemenea, colegilor Alex Eftimie și Andrei Faur pentru contribuția adusă, și colegilor Alex Juncu, Lucian Grijincu, Călin Iorgulescu, Voichița Iancu, Andrei Dumitru, Laura Gheorghe pentru revizuirea materialului pe parcursul finalizării acestuia.

Adresăm mulțumiri speciale echipei cursului de Utilizarea Sistemelor de Operare care ne-a oferit o atmosferă de suport, implicare și energie pentru realizarea cărții. Forma actuală a cărții se bazează pe efortul susținut depus de-a lungul numeroaselor activități din jurul cursului de Utilizarea Sistemelor de Operare.

Nu în ultimul rând, mulțumim cititorilor, la primul pas în domeniul plin de provocări și de satisfacții al calculatoarelor. Ei sunt cei cărora le dedicăm această carte. Ne-a făcut plăcere să o scriem, și sperăm că măcar o parte din entuziasmul nostru să se convertească în pasiune pentru cititorii acestor pagini.

Autorii

Cuprins

1 Introducere **1**

1.1 Ce este un sistem de operare? 1
1.1.1 Funcțiile unui sistem de operare 3
1.1.2 Tipuri de sisteme de operare 4
1.1.3 Functionarea sistemelor de operare 5
1.2 Scurt istoric al sistemelor de operare 7
1.2.1 Prima generație 7
1.2.2 A doua generație 8
1.2.3 A treia generație 8
1.2.4 A patra generație 9
1.3 Sisteme de operare moderne 9
1.3.1 Sisteme de operare desktop și server 10
1.3.2 Familia Windows 11
1.3.3 Mac OS X 11
1.3.4 Linux 12
1.3.5 Alte Unix-uri 13
1.3.6 Comunități open source 14
1.3.7 Sisteme de operare pentru dispozitive mobile 15
1.4 Studiu de caz 16
1.4.1 Virtualizare. Rularea unui SO dintr-o mașină virtuală 16

2 Instalarea Linux. Configurări de bază **23**

2.1 Linux. Distribuții Linux 23
2.1.1 Principalele distribuții Linux 24
2.1.2 Debian. Ubuntu 25
2.1.3 Tipuri de distribuții Linux 27
2.2 Instalarea Linux 28
2.2.1 Alegerea distribuției Linux 28
2.2.2 Pregătirea sistemului 29
2.2.3 Live CD 31
2.2.4 Instalare Kubuntu 32
2.3 Interacțiunea cu sistemul de operare 38
2.3.1 Interfețele cu utilizatorul 38
2.3.2 Interfața în linie de comandă (CLI) 40
2.3.3 Oprirea sistemului de calcul 44
2.4 Configurări de bază ale SO 45
2.4.1 Administrarea sistemului Linux 45
2.4.2 Asigurarea conectivității la Internet 47
2.4.3 Actualizarea sistemului și a pachetelor 47
2.4.4 Administrarea utilizatorilor și a grupurilor de utilizatori 48

2.5 Studii de caz 48
2.5.1 Interoperabilitatea sistemului de fișiere între Linux și Windows . 48
2.5.2 GParted 49

3 Gestiunea pachetelor și utilizatorilor 53
3.1 Gestiunea utilizatorilor 53
3.1.1 UID, GID 55
3.1.2 Adăugarea și ștergerea utilizatorilor 55
3.1.3 Adăugarea și ștergerea unui grup de utilizatori 56
3.1.4 Modificarea datelor unui utilizator 56
3.1.5 Adăugarea și ștergerea utilizatorilor 57
3.2 Gestiunea pachetelor 58
3.3 Studii de caz 64
3.3.1 Fișierele în care sunt stocate informații despre utilizatori 64
3.3.2 Actualizarea unui sistem Debian sau Ubuntu 66

4 Sisteme de fișiere 69
4.1 Noțiuni introductive 69
4.1.1 Ce este un sistem de fișiere 69
4.1.2 Ierarhia sistemului de fișiere 70
4.1.3 Căi relative și căi absolute 72
4.1.4 Variabila de mediu PATH 74
4.2 Tipuri de fișiere 74
4.2.1 Terminologie 74
4.2.2 Detecția tipului fișierelor 75
4.3 Operații uzuale asupra fișierelor și directoarelor 76
4.3.1 Afișarea și schimbarea directorului curent 76
4.3.2 Afișarea conținutului fișierelor 77
4.3.3 Listarea conținutului unui director 77
4.3.4 Crearea fișierelor/directoarelor 79
4.3.5 Copiere/mutare/redenumire/ștergere 80
4.3.6 Arhivarea fișierelor și dezarhivarea 82
4.3.7 Backup 85
4.4 Redirectări de comenzi 85
4.4.1 Descriptorii de fișier 85
4.4.2 Redirectări 86
4.5 Drepturi de acces 87
4.5.1 Utilizatori și grupuri de utilizatori vs. liste de acces 88
4.5.2 Clasificarea drepturilor de acces 89
4.5.3 Vizualizarea drepturilor de acces 89
4.6 Căutarea fișierelor 90
4.6.1 Comanda find 90
4.6.2 Comanda locate 91
4.6.3 Comanda whereis 91
4.6.4 Comanda which 92
4.6.5 Comanda type 92
4.7 Tipuri de sisteme de fișiere 92
4.7.1 Integritatea datelor 93
4.7.2 Alegerea unui sistem de fișiere 93
4.7.3 Adresarea într-un sistem de fișiere 95

4.8 Lucrul cu sistemele de fișiere . . . 95
4.8.1 Crearea unui sistem de fișiere . . . 96
4.8.2 Montarea unui sistem de fișiere . . . 97
4.8.3 Repararea unui sistem de fișiere . . . 100
4.8.4 Crearea unei imagini pentru un sistem de fișiere . . . 100
4.8.5 "Ştergerea" unui sistem de fișiere . . . 101
4.8.6 Monitorizarea utilizării discului . . . 101
4.9 Tendințe în sistemele de fișiere . . . 102
4.9.1 Sisteme de fișiere în userspace . . . 102
4.9.2 ZFS, ext4, btrfs . . . 104
4.10 Studii de caz . . . 104
4.10.1 Comenzi pentru lucrul cu fișiere în Windows . . . 104
4.10.2 Utilizarea în siguranță a comenzilor pe fișiere . . . 104
4.10.3 Montarea unui sistem de fișiere FAT32 astfel încât toți utilizatorii să aibă drept de scriere pe el . . . 105
4.10.4 Montarea unui sistem de fișiere FAT32 astfel încât utilizatorii dintr-un anumit grup să aibă drept de scriere pe acesta . . . 106
4.10.5 Sistem de fișiere într-un fișier . . . 106
4.10.6 ntfs-3g . . . 107

5 Procese **111**
5.1 Noțiuni introductive . . . 111
5.1.1 Ce este un proces? . . . 112
5.1.2 Deosebirea dintre un proces și un program . . . 112
5.1.3 Structura unui proces . . . 113
5.1.4 Stările unui proces. Multiprogramare și multiprocesare . . . 115
5.2 Ierarhia de procese în Unix. Vizualizarea proceselor sistemului . . . 116
5.2.1 Utilitarul ps . . . 117
5.2.2 Utilitarul pstree . . . 122
5.2.3 Utilitarul pgrep . . . 123
5.2.4 Utilitarul top . . . 124
5.2.5 Timpul de execuție al unui proces. Comanda time . . . 126
5.2.6 Sistemul de fișiere procfs . . . 127
5.3 Rularea proceselor în background. Job-uri. Daemoni . . . 130
5.3.1 Rularea unui proces în background . . . 130
5.3.2 Suspendarea unui proces . . . 131
5.3.3 Controlul job-urilor . . . 131
5.3.4 Daemoni . . . 133
5.4 Semnale . . . 135
5.4.1 Semnale importante UNIX . . . 136
5.4.2 Comenzile kill, killall și pkill . . . 137
5.4.3 Transmiterea de semnale prin combinații de taste . . . 139
5.5 Comunicația între procese . . . 139
5.5.1 Operatorul | (pipe) . . . 140
5.6 Swapping . . . 141
5.7 Studii de caz . . . 142
5.7.1 Managementul proceselor/serviciilor pe Windows . . . 142
5.7.2 Procese importante . . . 143
5.7.3 Prioritatea unui proces . . . 145

6 Pornirea și inițializarea sistemului **151**
6.1 Pornirea sistemului . . . 151
6.1.1 Problematica pornirii sistemului – bootstrapping . . . 152
6.1.2 Etapele pornirii sistemului . . . 152
6.1.3 BIOS . . . 153
6.1.4 POST . . . 154
6.2 Bootloader . . . 154
6.2.1 Dispozitive boot-abile . . . 155
6.2.2 Structura sectorului de boot pentru un dispozitiv boot-abil . . . 156
6.2.3 Mecanismul de funcționare a unui bootloader . . . 158
6.2.4 GRUB . . . 159
6.3 Configurarea GRUB . . . 162
6.4 Încărcarea nucleului . . . 165
6.4.1 Imaginea de nucleu . . . 165
6.4.2 Opțiuni de boot-are pentru nucleu . . . 166
6.5 Inițializarea sistemului . . . 167
6.5.1 init . . . 167
6.5.2 upstart . . . 171
6.5.3 getty și login . . . 173
6.5.4 Sesiune de shell . . . 174
6.6 Studiu de caz . . . 175
6.6.1 Grub 2 . . . 175
6.6.2 Pornirea sistemului de operare Windows XP . . . 177
6.6.3 Interoperabilitate Linux-Windows . . . 179
6.6.4 Personalizarea ecranului GRUB . . . 180

7 Analiza hardware a sistemului **185**
7.1 Structura unui sistem de operare . . . 185
7.2 Considerente hardware . . . 186
7.2.1 Structura unui sistem de calcul . . . 187
7.2.2 Procesorul . . . 188
7.2.3 Ierarhia de memorie . . . 188
7.2.4 Placa de bază; interconectarea componentelor . . . 191
7.2.5 Dispozitive periferice; magistrale . . . 192
7.3 Suportul pentru dispozitive la nivelul kernel-ului . . . 193
7.3.1 Listarea modulelor încărcate la un moment dat în sistem . . . 194
7.3.2 Încărcarea unui modul . . . 195
7.3.3 Descărcarea unui modul din kernel . . . 195
7.4 Analiza hardware a unui sistem (magistrale, chipset, CPU, memorie, dispozitive) . . . 196
7.4.1 Lista hardware – /sys . . . 196
7.4.2 Comenzi pentru afișarea dispozitivelor . . . 196
7.4.3 Monitorizarea stării dispozitivelor hardware . . . 199
7.5 Interfața cu dispozitivele din userspace – **udev** și `/dev` . . . 201
7.6 Lucrul cu dispozitive (`/dev`) . . . 202
7.6.1 Tipuri de dispozitive . . . 203
7.6.2 Întreruperi hardware . . . 204
7.6.3 Adrese I/O . . . 205
7.6.4 Adrese DMA . . . 206

7.6.5 Manipularea datelor la nivel scăzut (comanda dd) 206
7.7 Analiza sistemului 208
7.7.1 Informații despre sistem 208
7.7.2 Kernel "tunables" 210
7.8 Studii de caz 211
7.8.1 Salvarea și restaurarea MBR și a tabelei extinse de partiții . . . 211
7.8.2 Salvarea conținutului unui disc cu sectoare inaccesibile 212
7.8.3 Crearea unei imagini de CD; montarea unei imagini 212
7.8.4 Utilizarea unui fișier de pe o partiție FAT32 ca fișier de swap pentru un LiveCD Linux 213

8 Configurări de rețea 217
8.1 Concepte de rețea 217
8.1.1 Noțiuni de bază 218
8.1.2 IPv6 219
8.1.3 Moduri de adresare: adresare unicast, multicast, broadcast . . 219
8.2 Parametri de rețea 220
8.2.1 Adresă IP și mască de rețea 220
8.2.2 Ruter implicit (default gateway) 222
8.2.3 DNS 223
8.3 Configurări temporare 225
8.3.1 Interfețe de rețea. Configurări permanente și configurări temporare 225
8.3.2 Configurarea temporară statică a unei adrese IP pe o interfață . 226
8.3.3 Asigurarea conectivității la Internet. Configurarea temporară statică a unei rute implicite 229
8.4 Configurări permanente 230
8.4.1 Configurarea permanentă a unei interfețe de rețea 230
8.4.2 Configurarea permanentă statică a unei rute implicite 232
8.4.3 DHCP. Configurarea unei interfețe în mod automat 232
8.4.4 DNS. Configurarea serverelor de DNS 233
8.4.5 Aliasuri. /etc/hosts 235
8.4.6 Configurarea numelui stației curente 235
8.5 Testarea configurațiilor de rețea 236
8.5.1 Ping 236
8.5.2 Traceroute 237
8.5.3 Cum se depistează problemele uzuale în cazul configurărilor de rețea 238
8.6 Studii de caz 240
8.6.1 Verificarea în linie de comandă a parametrilor de rețea în Windows 240
8.6.2 Configurări de rețea în Windows Vista 241
8.6.3 Configurarea PPPoE în Linux 243

9 Servicii de rețea 249
9.1 Concepte specifice aplicațiilor de rețea 249
9.1.1 Stiva TCP/IP 249
9.1.2 Implementarea parțială a stivei TCP/IP 252
9.1.3 Modelul client-server 254
9.1.4 Porturi 255

9.2 Execuția comenzilor la distanță . . . 257
9.2.1 Telnet . . . 257
9.2.2 SSH . . . 258
9.2.3 Utilitarul wget . . . 262
9.3 Email – Poșta electronică . . . 263
9.3.1 Arhitectură și funcționare . . . 264
9.3.2 Clienți de email . . . 266
9.3.3 Securitatea serviciului de email . . . 267
9.4 WWW . . . 268
9.4.1 Tehnologiile de bază ale WWW . . . 269
9.4.2 Funcționarea serviciului . . . 272
9.4.3 Servere Web . . . 272
9.4.4 Clienții Web . . . 273
9.5 Studii de caz . . . 273
9.5.1 Utilitarul cURL . . . 273
9.5.2 FTP . . . 274

10 Elemente de securitate **279**
10.1 Problematica securității . . . 279
10.1.1 Principii de bază . . . 280
10.1.2 Termeni . . . 283
10.2 Securizarea sistemului . . . 284
10.2.1 Securitatea sistemului de operare . . . 284
10.2.2 Controlul accesului . . . 286
10.2.3 Parole . . . 287
10.2.4 Securitatea sistemului de fișiere . . . 289
10.3 Întreținerea sistemului . . . 293
10.3.1 Monitorizarea sistemului . . . 293
10.3.2 Jurnalizarea și gestiunea jurnalelor . . . 296
10.3.3 Limitarea drepturilor . . . 298
10.4 Atacuri de rețea . . . 303
10.4.1 Tipuri de atacuri în rețea . . . 303
10.4.2 Viruși, viermi, troieni . . . 305
10.4.3 Scanarea porturilor . . . 307
10.5 Securizarea rețelei . . . 309
10.5.1 Firewall-uri . . . 310
10.5.2 Criptarea informației . . . 310
10.5.3 Monitorizarea rețelei . . . 312
10.6 Studiu de caz . . . 314
10.6.1 Drepturile pe fișiere în NTFS . . . 314
10.6.2 Recuperarea parolei . . . 315

11 Compilare și linking **319**
11.1 Introducere . . . 319
11.1.1 Editoare . . . 320
11.1.2 Compilare și interpretare . . . 321
11.1.3 De la sursă la executabil, de la executabil la proces . . . 322
11.1.4 Pachete necesare . . . 323
11.2 Compilare. GCC . . . 324
11.2.1 Utilizare GCC . . . 325

11.2.2 Compilarea din surse multiple . . . 327
11.3 Etapele compilării (inclusiv link-editarea) . . . 328
11.3.1 Preprocesarea . . . 330
11.3.2 Compilarea . . . 331
11.3.3 Asamblarea . . . 333
11.3.4 Optimizarea compilării . . . 335
11.3.5 Link-editarea . . . 337
11.3.6 Fișiere executabile . . . 339
11.4 Biblioteci de funcții . . . 341
11.4.1 Tipuri de biblioteci . . . 342
11.4.2 Informații despre bibliotecile de funcții . . . 343
11.4.3 Utilizarea bibliotecilor . . . 344
11.5 Automatizarea sarcinilor – make . . . 345
11.5.1 Cel mai simplu Makefile . . . 346
11.5.2 Folosirea dependențelor . . . 347
11.5.3 Dependențe ierarhice . . . 347
11.5.4 Target-ul clean . . . 350
11.5.5 Target-urile .PHONY și all . . . 350
11.5.6 Variabile în Makefile . . . 352
11.5.7 Sintaxă Makefile . . . 352
11.5.8 Moduri de utilizare a Make . . . 353
11.6 Portabilitate . . . 354
11.6.1 Portabilitatea la nivelul arhitecturii sistemului de calcul . . . 354
11.6.2 Portabilitatea unui limbaj de programare . . . 354
11.6.3 Portabilitatea la nivelul sistemului de operare . . . 356
11.7 Studiu de caz . . . 356
11.7.1 GCC în Windows . . . 356
11.7.2 Link-editarea modulelor C și a modulelor C++ . . . 357

12 Shell scripting **365**
12.1 Noțiuni introductive . . . 365
12.1.1 De ce shell scripting? . . . 366
12.1.2 Facilități oferite de scripturile shell . . . 367
12.2 Interacțiunea cu shell-ul . . . 368
12.2.1 Editarea comenzilor . . . 368
12.2.2 Folosirea istoricului. Completare automată . . . 369
12.2.3 Comenzi interne (*built-in*) și comenzi externe . . . 371
12.2.4 Execuția unei comenzi shell . . . 373
12.3 Scripturi shell . . . 374
12.3.1 Cel mai simplu script shell . . . 374
12.3.2 Comentarii într-un script shell . . . 376
12.3.3 Comenzi simple shell . . . 376
12.3.4 One liners . . . 378
12.4 Programarea shell . . . 384
12.4.1 Variabile . . . 384
12.4.2 Caractere speciale shell . . . 387
12.4.3 Instrucțiuni de decizie . . . 391
12.4.4 Cicluri în shell . . . 395
12.5 Filtre de text . . . 399

12.5.1 cat, tac, nl . . . 400
12.5.2 sort, uniq . . . 400
12.5.3 head, tail . . . 402
12.5.4 cut . . . 403
12.5.5 tr . . . 404
12.5.6 wc . . . 405
12.5.7 grep . . . 405
12.5.8 sed . . . 408
12.5.9 awk . . . 410
12.6 Comenzi de lucru cu fișiere . . . 414
12.6.1 xargs . . . 414
12.6.2 locate . . . 415
12.6.3 find . . . 416
12.7 Expandarea în shell . . . 419
12.7.1 Simbolul $ . . . 419
12.7.2 Expresii regulate în shell . . . 421
12.8 Parametrii unui script shell . . . 421
12.8.1 Comanda shift . . . 422
12.8.2 Parametri speciali . . . 423
12.8.3 Exemplu de utilizare a parametrilor . . . 423
12.9 Funcții . . . 424
12.9.1 Sintaxa unei funcții . . . 424
12.9.2 Parametrii unei funcții . . . 425
12.10 Scripturile de pornire Bash . . . 426
12.10.1 Variabile de mediu . . . 427
12.11 Studii de caz . . . 428
12.11.1 Contorizarea numărului de utilizatori autentificați în sistem . . . 428
12.11.2 Schimbarea promptului shell . . . 429
12.11.3 Batch scripting în Windows . . . 431
12.11.4 PowerShell pe Windows . . . 433

13 Mediul grafic 441
13.1 Concepte în mediul grafic . . . 442
13.1.1 Tipuri de imagini . . . 442
13.1.2 Fonturi. Unicode . . . 443
13.2 Interfața grafică în Linux. Componente . . . 444
13.2.1 X Window System . . . 444
13.2.2 Arhitectura X Window System . . . 445
13.3 Pornirea și oprirea interfeței grafice . . . 448
13.4 Configurarea serverului X . . . 450
13.4.1 Configurarea rezoluției . . . 450
13.4.2 Configurarea tastaturii . . . 451
13.5 Configurarea sistemului din KDE . . . 452
13.5.1 System Settings . . . 452
13.5.2 Schimbarea aspectului interfeței grafice . . . 452
13.5.3 Configurări de bază ale sistemului de operare . . . 453
13.5.4 Configurări administrative . . . 454
13.5.5 Configurarea rețelei . . . 454
13.5.6 Managementul utilizatorilor . . . 454

13.5.7 Monitorizarea sistemului și managementul proceselor 455
13.5.8 Gestiunea pachetelor 457
13.6 Servicii desktop 458
13.6.1 Pornirea facilă a aplicațiilor 458
13.6.2 Căutarea fișierelor în sistem 459
13.6.3 Notificări 460
13.7 Aplicații KDE vs. GNOME 460
13.8 Studii de caz 460
13.8.1 Configurarea X peste SSH 460
13.8.2 Instalarea și configurarea VNC 462
13.8.3 Remote Desktop Connection 464

14 Utilitare pentru dezvoltare **469**
14.1 Introducere 469
14.2 Coding style (indent, astyle) 470
14.3 Editorul Vim 473
14.4 Sisteme de control al versiunii 479
14.4.1 Principii 479
14.4.2 Subversion 480
14.4.3 Git 481
14.5 Analiza și parcurgerea codului 481
14.5.1 ctags 482
14.5.2 Analiza statică a codului – splint 482
14.6 Automatizarea compilării 484
14.6.1 Compilarea unei aplicații din surse 484
14.7 Execuția unui program 485
14.7.1 Zonele de memorie ale unui executabil și proces 485
14.7.2 Utilizarea bibliotecilor partajate 486
14.7.3 Analiza apelurilor de sistem și a semnalelor 487
14.8 Depanarea unui program 489
14.8.1 gdb, ddd 489
14.8.2 valgrind 492
14.9 Medii integrate de dezvoltare 494
14.9.1 Eclipse 494
14.9.2 Anjuta 494
14.10 Managementul proiectelor software 496
14.10.1 Procese de dezvoltare software 496
14.10.2 Aplicații web pentru managementul proiectelor software 497
14.10.3 Resurse online pentru dezvoltarea proiectelor 498
14.11 Resurse de documentare pentru dezvoltator 498
14.11.1 Documentație oficială 498
14.11.2 Documentarea programelor proprii 499
14.11.3 Cărți și tutoriale 500
14.11.4 Documentație din Internet 501
14.12 Studiu de caz 502
14.12.1 Microsoft Visual Studio 502

15 Viața în Linux **507**
15.1 Muzica în Linux 507
15.2 Filme în Linux 509

15.3 Scrierea unui CD/DVD în Linux 510
15.4 Messenger în Linux 511
15.5 BitTorrent în Linux 512
15.6 Imprimanta în Linux 514
15.7 Jocuri în Linux 515
15.8 Dual-monitor în Linux 516
15.9 Documente office în Linux 518

A Răspunsuri la întrebări **521**

Bibliografie **529**

Glosar **531**

Abrevieri

ACL – Access Control List
ACPI – Advanced Configuration and Power Interface
AGP – Advanced Graphics Port
API – Application Programming Interface
BIOS – Basic Input Output System
CLI – Command Line Interface
CMOS – Complementary Metal Oxide Semiconductor
CPU – Central Processing Unit
DDD – Data Display Debugger
DE – Desktop Environment
DHCP – Dynamic Host Configuration Protocol
DLL – Dynamic-link library
DM – Display Manager
DMA – Direct Memory Access
DNS – Domain Name System
DoS – Denial of Service
DRAM – Dynamic RAM
ELF – Executable and Linking Format
FSB – Front Side Bus
FTP – File Transfer Protocol
FUSE – Filesystem in Userspace
GCC – GNU Compiler Collection
GDB – GNU Debugger
GID – Group Identifier
GNU – GNU's Not Unix
GNU CPP – GNU C preprocessor
GUI – Graphical User Interface
HDD – hard disk drive
HTML – Hypertext Markup Language
HTTP – Hypertext Transfer Protocol
ICH – I/O Controller Hub
ICMP – Internet Control Message Protocol
IDE – Integrated Development Environment
IDS – Intrusion Detection System
IMAP – Internet Mail Access Protocol
IP – Internet Protocol
IRC – Internet Relay Chat
IRQ – interrupt request
ISP – Internet Service Provider

LAN – Local Area Network
LTS – Long Time Support
LVM – Logical Volume Manager
MAN – Metropolitan Area Network
MBR – Master Boot Record
MCH – Memory Controller Hub
MIME – Multipuprose Internet Mail Extensions
MinGW – Minimalist GNU for Windows
MSVC – Microsoft Visual C++
NAT – Network Address Translation
PCI – Peripheral Component Interconnection
PID – process id
PKI – Public Key Infrastructure
POP3 – Post Office Protocol
POSIX – Portable Operating System Interface
POST – Power-on Self Test
PPP – Point to Point Protocol
PPPoE – PPP over Ethernet
RAID – Redundant Array of Inexpensive Disks
RAM – Random Access Memory
RDC – Remote Desktop Connection
RPM – Rotations Per Minute
RTOS – sisteme de operare în timp real
S.M.A.R.T. – Self-Monitoring, Analysis and Reporting Technology
SMTP – Simple Mail Transfer Protocol
SNMP – Simple Network Management Protocol
SO – sistem de operare
SRAM – Static RAM
SSH – Secure Shell
SVN – Subversion
TCP – Transmission Control Protocol
TFTP – Trivial File Transfer Protocol
TLD – Top Level Domain
TTL – Time To Live
TUI – Text User Interface
UDP – User Datagram Protocol
UID – User Identifier
URL – Uniform Resource Locator
USB – Universal Serial Bus
VBR – Volume Boot Record
VCS – Version Control System
VNC – Virtual Network Computing
WAN – Wide Area Network
WIMP – window, icon, menu, pointing device
WLAN – Wireless LAN
WM – Window Manager
WWW – World Wide Web
XHTML – Extensible Hypertext Markup Language

Capitolul 1

Introducere

They have computers, and they may have other weapons of mass destruction.

Janet Reno

Ce se învață din acest capitol?

- Ce este un sistem de operare
- Funcțiile unui sistem de operare
- Tipuri de sisteme de operare
- Istoricul sistemelor de operare
- Sisteme de operare moderne
- Windows, Mac OS X, Linux
- Virtualizare

1.1 Ce este un sistem de operare?

Sistemele de operare reprezintă un concept familiar tuturor utilizatorilor de calculatoare personale. Ele apar tot mai des și în implementarea de funcționalități din alte aparate, de la telefoanele mobile la cardurile bancare inteligente. Există însă și dispozitive care nu au (încă) un sistem de operare, în principal deoarece execută un număr mic de operațiuni care nu necesită nicio modificare. De exemplu, calculatorul dintr-un cuptor cu microunde execută un program simplu fixat prin circuite fizice. Sistemele de operare sunt utile în cazul calculatoarelor care trebuie să se adapteze unui mediu în schimbare, ele permițând modificarea comportamentului calculatorului fără schimbarea succesiunii operațiilor hardware.

Un **sistem de operare** (abreviat în continuare *SO*) este un set de programe care controlează distribuția resurselor unui calculator și mediază comunicarea dintre hardware, pe de o parte, și aplicațiile utilizatorilor, pe de altă parte.

Scopul unui SO este să asigure stabilitate și flexibilitate în funcționarea calculatorului. Sistemele de operare permit, în general, interacțiunea cu mai mulți utilizatori, rularea mai multor aplicații, și modificarea strategiilor de răspuns ale calculatorului la o anumită problemă.

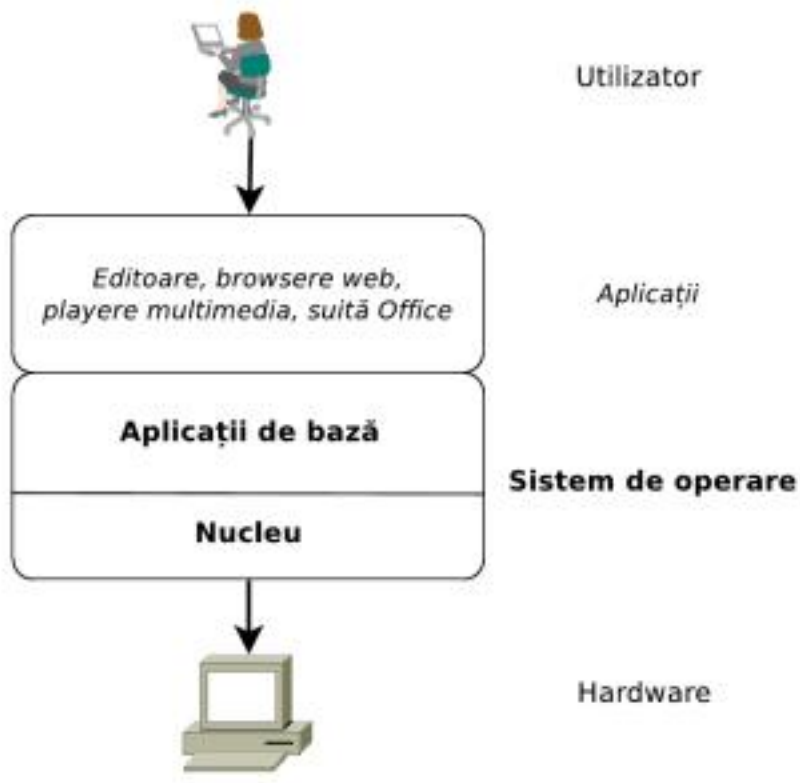


Figura 1.1: Rolul unui sistem de operare

În figura 1.1 este prezentat rolul de mediere al sistemului de operare între utilizatori și hardware. Sistemul de operare va avea acces direct la resursele hardware, în vreme ce utilizatorii folosesc cel mai adesea aplicații ce apelează prin intermediul bibliotecilor sau direct funcții ale sistemului de operare.

Componenta esențială a sistemului de operare o reprezintă **nucleul** sau **kernelul**. Acesta are rolul de gestiune a resurselor fizice și de oferire a unei interfețe comune aplicațiilor. Nucleul nu este o aplicație în sine, ci este un mediu în cadrul căruia pot rula alte aplicații.

Programele de bază sunt aplicațiile fundamentale ale unui sistem de operare care permit interacțiunea cu nucleul și cu sistemul fizic.

Astfel de aplicații sunt interpretoarele de comenzi, utilitare de gestiune a utilizatorilor și a sistemului de fișiere, biblioteci etc. În general nu este bine conturată definiția de aplicație de bază; se consideră că o aplicație este o aplicație de bază atunci când oferă servicii pentru o bună parte din celelalte componente ale sistemului.

Din punct de vedere *strict tehnic* sistemul de operare este însuși nucleul. Din perspectiva utilizatorului sitemul de operare este compus din nucleu și suita programelor de bază. Programele de bază sunt acele aplicații ce permit utilizatorului interacțiunea cu resursele hardware ale sistemului, această relație fiind ilustrată în figura 1.1.

Comparativ, un calculator poate fi privit ca un aeroport, iar sistemul de operare ca turnul de control care coordonează traficul. Resursele aeroportului sunt reprezentate de pistele și avioanele pe care le pune la dispoziție. Fără un turn de control care să asigure gestiunea și organizarea acestor resurse, pe un aeroport nu s-ar putea evita

întârzierile și coliziunile. În cazul calculatoarelor, rolul unui sistem de operare este de a coordona traficul și accesul la resurse, întocmai precum turnul de control al unui aeroport.

1.1.1 Funcțiile unui sistem de operare

O primă funcție a unui sistem de operare constă în **administrarea resurselor hardware ale sistemului**. În cazul unui calculator, este vorba mai ales despre accesul la procesor, memorie, hard-disk, comunicarea în rețea, precum și cu alte dispozitivele de intrare/ieșire. Aceste resurse pot fi solicitate de aplicații multiple și de utilizatori multipli.

Sistemul de operare este la rândul său o aplicație, care necesită resurse (și, implicit, accesul la acestea) pentru a funcționa. Prin urmare, acesta trebuie să își asigure accesul privilegiat la resurse. Un SO trebuie să mențină **controlul asupra calculatorului** și să facă posibilă utilizarea sa predictibilă și cât mai eficientă.

Sistemul de operare oferă o **abstractizare a resurselor hardware**. Practic SO este o extensie a sistemului fizic, oferind o inferfață simplificată folosită pentru medierea accesului utilizatorilor și aplicațiilor la resursele hardware.

Sistemul de operare este responsabil cu oferirea unui set limitat de informații și funcții de lucru cu hardware-ul, în loc de descrierea exactă a ceea ce se întâmplă în hardware. Practic, SO oferă dispozitivelor și aplicațiilor (și, în ultimă instanță, utilizatorilor) o **transparență față de modul de funcționare a hardware-ului**.

În ceea ce privește comunicarea dintre hardware și aplicații, sistemul de operare trebuie să creeze o identitate calculatorului, care să rămână relativ stabilă în dialogul acestuia cu aplicații diverse, chiar dacă au loc upgrade-uri de hardware. Sistemul de operare are deci rolul de **asigurare a securității și integrității sistemului de calcul**. Sistemul de operare va controla accesul utilizatorilor și proceselor la resursele hardware ale sistemului și va preveni execuția de instrucțiuni invalide, malițioase sau privilegiate.

Sistemul de operare este și cel care permite unei aplicații să ruleze pe sisteme fizice diferite, deoarece uniformizează răspunsurile date de calculator și intermediază accesul la resurse. Sistemele de operare trebuie prin urmare să poată interacționa cu arhitecturi hardware și cu periferice foarte diverse, el fiind cel care facilitează accesul aplicațiilor la resursele hardware. Această funcție se numește **portabilitate** și se referă la oferirea unei interfețe unice pentru utilizator, indiferent de diferențele hardware dintre arhitecturi.

Un SO folosește **drivere** pentru comunicarea cu dispozitivele periferice.

> **Driverele** sunt programe care traduc comenzile sistemului de operare în comenzi inteligibile echipamentelor, precum și ieșirea acestora în mesaje accesibile sistemului.

Rolul SO a fost extins prin punerea la dispoziția aplicațiilor a unei colecții de biblioteci (funcții) pe care programele le pot folosi pentru a rezolva probleme specifice legate de interacțiunea cu componentele calculatorului.

Sistemul de operare va include utilitare de diagnosticare și monitorizare a funcționării diferitelor componente atât hardware cât și software. Un sistem de operare poate,

desigur, să eșueze în realizarea funcțiilor sale. De exemplu, un tip foarte frecvent de eroare constă în blocarea funcționării calculatorului. Un astfel de "crash" poate fi cauzat de unele imperfecțiuni ale sistemului, dar poate fi favorizat și de funcționarea necorespunzătoare a anumitor aplicații. Un al doilea tip frecvent de eroare se referă la problemele de securitate, care pot permite intruziunea unui utilizator sau a unei aplicații malițioase. De asemenea, este posibil ca un sistem de operare să nu reușească să interacționeze corect cu anumite periferice sau echipamente. În principal, funcționarea corectă a unui sistem de operare poate fi păstrată prin realizarea metodică a unui număr limitat de operații:

- repararea problemelor (bug-urilor) din sistemul de operare însuși;
- repararea bug-urilor din aplicații;
- eliminarea virușilor, viermilor, a spyware etc;
- instalarea de versiuni actualizate ale sistemului de operare și aplicațiilor acestuia.

1.1.2 Tipuri de sisteme de operare

Există diverse criterii de clasificare a sistemelor de operare. Clasificarea de mai jos este realizată ținând cont de sistemele fizice pe care rulează și domeniul de utilizare.

Sistemele de operare de pe smart carduri sunt cele mai simple, servind deseori funcții reduse (precum în cazul cărților de debit) și administrând resurse foarte limitate. Totuși, ele oferă un management de bază al acestor resurse, inclusiv a spațiului de stocare, putând oferi și servicii de securitate/criptare a datelor ce le dețin.

Sistemele de operare embedded sunt incluse în dispozitive precum televizoarele sau telefoanele mobile, având constrângeri specifice funcționalității lor. Sistemele de operare pentru dispozitivele embedded variază de la sisteme dedicate unui singur scop, care nu oferă o interfață de interacțiune utilizatorului, până la sisteme multitasking (precum în cazul sistemelor de operare pentru PDA) care oferă multe dintre fucționalitățile sistemelor de operare pentru calculatoarele personale. **Sistemele de operare în timp real** (RTOS – Real Time Operating Systems) sunt folosite mai ales în echipamente științifice, industriale și de precizie. Scopul principal al acestora este să optimizeze utilizarea resurselor echipamentelor și să asigure predictibilitatea funcționării lor, dat fiind că eșecurile în coordonarea subcomponentelor pot avea consecințe dezastruoase.

Timpul este o variabilă esențială, deoarece anumite sarcini trebuie realizate într-un interval precis pentru a se putea coordona cu alte sarcini. Astfel de sisteme de operare nu sunt în general menite să permită comunicarea cu utilizatorul, dat fiind că nu este necesar să fie modificate, ele fiind concepute strict pentru executarea anumitor procese tehnologice.

RTOS pot fi împărțite în sisteme în timp real hard, prevalente în procesele industriale, în care constrângerile temporale sunt întotdeauna prioritare, și sisteme în timp real soft, precum sistemele dedicate multimedia, în care discordanțe temporale minore sunt acceptabile sau insesizabile. Multe din sistemele de operare moderne oferă funcționalități specifice sistemelor de operare în timp real soft.

Există numeroase sisteme de operare de timp real. Exemple sunt RTLinux[1], VxWorks[2], TRON[3] și QNX[4]. Majoritatea sistemelor de operare de timp real sunt folosite în cadrul dispozitivelor embedded. Nu există o linie de demarcație clară între sisteme de operare embedded și sisteme de operare în timp real. Sistemele de operare moderne sunt **sisteme multi-tasking** (permit execuția simultană a mai multor sarcini) și **multi-user** (mai mulți utilizatori pot folosi în același timp resursele sistemului). Sistemele multi-user urmăresc în principal să optimizeze funcționarea calculatorului prin echilibrarea cerințelor utilizatorilor și izolarea problemelor acestora. Există sisteme de operare specializate pentru situațiile de conectare a calculatoarelor în rețea, precum și sisteme de operare specializate pentru aplicațiile care rulează pe un server, deși aceste granițe tind să se estompeze.

În această categorie sunt incluse sistemele de operare precum cele din familiile Microsoft Windows, Unix și Apple Mac OS X, ce au ca scop optimizarea folosirii resurselor atunci când un utilizator rulează concomitent mai multe aplicații sau când mai mulți utilizatori interacționează cu un calculator în același timp. Aceste sisteme de operare sunt cele mai cunoscute de utilizatorii de PC-uri.

O practică uzuală pentru obținerea unei puteri sporite de procesare o reprezintă instalarea de procesoare multiple într-un singur sistem. Pentru managementul eficient al acestora și pentru a se putea folosi întreaga putere de procesare disponibilă, există **sisteme de operare pentru arhitecturi multiprocesor** special concepute pentru procesarea paralelă (simultană). Acestea pot fi variante modificate ale sistemelor de operare pentru servere, optimizate pentru gestiunea paralelă a operațiilor pe care le execută.

O dată cu proliferarea sistemelor de operare multi-core, majoritatea sistemelor de operare contemporane pot fi folosite atât pe arhitecturi monoprocesor, cât și pe cele multiprocesor. Sisteme de operare precum Windows Vista, Windows 7, distribuțiile Linux bazate pe kernel 2.6 și Mac OS X oferă suport pentru sisteme multiprocesor.

1.1.3 Funcționarea sistemelor de operare

Un sistem de operare este responsabil de realizarea mai multor sarcini. Responsabilitățile principale sunt:

- administrarea procesorului;
- administrarea memoriei;
- administrarea echipamentelor și a perifericelor;
- administrarea sistemelor de stocare a datelor;
- interfața cu aplicațiile și interfața cu utilizatorii.

Rolul principal al sistemului de operare constă în impunerea de reguli pentru folosirea coerentă a resurselor (denumită și multiplexare). Aceasta se realizează fie prin

[1] http://www.rtlinuxfree.com/
[2] http://www.windriver.com/
[3] http://www.tron.org/index-e.html
[4] http://www.qnx.com/

alternarea în timp a accesului la resurse (multiplexare în timp), de exemplu în cazul administrării procesorului, fie prin delimitarea și alocarea unor segmente din resurse aplicațiilor ce le vor utiliza simultan (multiplexare în spațiu), de exmplu în cazul memoriei.

Responsabilitățile de mai sus sunt îndeplinite de nucleul sistemului de operare. Pe lângă acestea, sistemele de operare conțin componente și utilitare care permit realizarea unor sarcini precum:

- inițializarea și oprirea proceselor, inclusiv comunicarea între procese;
- organizarea aplicațiilor disponibile pe calculator;
- organizarea fișierelor în directoare;
- vizualizarea și editarea fișierelor, precum și redenumirea, copierea sau ștergerea fișierelor.

Administrarea procesorului

Administrarea procesorului vizează optimizarea accesului diferitelor unități software la resursele procesorului; practic, acest lucru se traduce în alocarea de cuante de timp pentru rularea unei aplicații pe un procesor.

Este important de observat că, deși sistemele de operare multi-tasking permit funcționarea simultană a mai multor aplicații, aceasta este de fapt simulată prin comutarea foarte rapidă de la o aplicație la alta a unui procesor care oferă doar facilități single-tasking. Execuția simultană în adevăratul sens al cuvântului are loc numai pe arhitecturi multiprocesor, unde fiecare procesor la un moment dat executa un proces.

Prin urmare, este esențial ca sistemul de operare să fie capabil de a răspunde la solicitările diferitelor aplicații. De asemenea, sistemul de operare trebuie să minimizeze proporția de timp în care procesorul este utilizat pentru execuția sarcinilor de întreținere, de comunicare internă etc, în favoarea timpului alocat cerințelor directe ale utilizatorului.

Elementele de bază cu care lucrează un sistem de operare (mai exact, elementele care folosesc procesorul) nu sunt aplicațiile, ci procesele și firele de execuție (threads).

Un proces (vezi capitolul 5) este un program care execută o acțiune și este controlat de utilizator, de o aplicație sau de sistemul de operare. O aplicație poate fi ea însăși un proces dar poate fi și inițiatoarea altor procese pentru a comunica cu alte calculatoare sau cu alte aplicații. Sistemele de operare inițiază numeroase procese pentru a administra resursele calculatorului, în majoritatea cazurilor, utilizatorii nu sunt conștienți de acest lucru.

Administrarea memoriei

Administrarea memoriei urmărește să asigure faptul că fiecare proces are suficientă memorie, fără să acceseze spațiul de memorie al altor procese. De asemenea, procesele trebuie să folosească adecvat diferitele tipuri de memorie existentă – ținând cont de accesibilitatea diferențiată a acestora.

Administrarea perifericelor

Administrarea perifericelor și a echipamentelor este realizată prin intermediul driverelor. Acestea sunt programe relativ autonome de sistemul de operare, fiind deseori furnizate de producătorul echipamentelor și instalate separat. Ele permit sistemului de operare să comunice cu echipamente noi fără să fie modificat. Comunicarea cu mediul este intermediată de spații tampon (buffere), în care datele de intrare sunt acumulate urmând să fie transmise procesorului în ritmul accesibil acestuia, evitând congestia atunci când apare o aglomerare datorită multitudinii de aplicații.

Interfața cu utilizatorul și aplicațiile

Graphical User Interface

Funcția de mediere a sistemului de operare are două subcomponente:

- interfața cu aplicațiile (API – *Application Programming Interface*) permite aplicațiilor să acceseze facil resursele hardware ale sistemului fără a controla toate detaliile tehnice ale aplicațiilor;
- interfața cu utilizatorul permite acestuia din urmă să comunice cu calculatorul, fie prin comenzi text (TUI – *Text User Interface*) sau printr-o interfață grafică (GUI – *Graphical User Interface*).

1.2 Scurt istoric al sistemelor de operare

Sistemele de operare pentru PC-urile actuale sunt nu numai rezultatul unei evoluții graduale, ci și al unei schimbări conceptuale. Dacă în urmă cu 30 de ani sistemele de operare dispuneau de funcționalități limitate, în zilele noastre sistemul de operare este acompaniat de interfața grafică și de aplicații diverse cu scopul de a fi cât mai accesibil utilizatorului individual neprofesionist. Sistemele de timp real și cele dedicate (embedded) sunt mai apropiate de formele inițiale, dezvoltate în anii '60-'70.

Din punct de vedere a evoluției sistemelor de operare pot fi distinse patru mari perioade, denumite și generații. Aceste generații se află în legătură strânsă cu dezvoltarea sistemelor fizice și a limbajelor de programare.

1.2.1 Prima generație

În prima perioadă, care s-a desfășurat în mare între anii 1945 și 1955, se dezvoltă prima generație de calculatoare, având la bază tehnologia tuburilor vidate. O inovație majoră este introducerea cartelelor perforate, ca modalitate de stocare a informației. În această etapă nu există niciun sistem de operare și nici măcar limbaje de programare; utilizatorul final este responsabil pentru interacțiunea directă cu hardware-ul.

1.2.2 A doua generație

Între anii 1955 și 1965 introducerea tranzistorului marchează trecerea la a doua generație de calculatoare, dominate de sistemele mainframe. Principalele limbaje utilizate sunt FORTRAN și limbajul de ansamblare.

Unitatea operațională principală de structurare a informației în interacțiunea cu calculatorul era **job-ul**, care consta într-unul sau mai multe programe scrise întâi pe hârtie și apoi introduse pe cartele.

O proporție substanțială din timpul de utilizare al mainframe-ului se pierde cu introducerea fizică a joburilor în calculator (prin realizarea și mutarea cartelelor). Pentru a reduce acest incovenient se dezvoltă tehnica procesării agregate a job-urilor (în batch-uri): mai multe job-uri sunt grupate într-un singur șir de cartele separate de delimitatori. Job-urile sunt ulterior imprimate pe bandă magnetică folosind un calculator relativ ieftin, apoi benzile sunt transmise mainframe-ului, urmând ca outputul să fie tipărit din nou prin intermediul calculatorului mai ieftin. Primul sistem de operare major bazat pe principiul "single stream batch processing" a fost dezvoltat de General Motors Research Laboratories pentru mainframe-ul IBM 701 în 1956.

1.2.3 A treia generație

A treia generație de calculatoare, bazată pe circuitele integrate, apare în perioada 1965-1980. Unul dintre cele mai importante inovații al acestei perioada este principiul utilizării în comun a timpului de procesor (timesharing). Conform acestui principiu, mai mulți utilizatori pot folosi simultan terminale conectate la același calculator.

În timesharing fiecare utilizator primește alternativ o fracțiune de timp, perioadă în care procesorul execută instrucțiunile acestuia. Astfel se valorifică eficient timpii morți datorați vitezei scăzute de acțiune a oamenilor față de cea a procesorului, reducând substanțial durata de elaborare și corectare a unui program.

Primul sistem bazat pe timesharing a fost CTSS – Compatible Time-Sharing System, realizat de MIT în 1961. Din sistemul de operare Multics, dezvoltat ulterior tot de MIT, au derivat primele variante de Unix, inițial sub conducerea lui Ken Thompson. Dennis Ritchie a elaborat limbajul C special pentru redactarea Unix, evitând astfel dependența de caracteristicile hardware, inerentă în folosirea limajului de ansamblare.

Unix a fost primul sistem de operare scris într-un limbaj de nivel superior, devenind astfel portabil de la un tip de calculator la altul. Datorită nevoii de standardizare a multiplelor variante de sisteme Unix care au apărut apoi independent pe baza codului sursă, IEEE a elaborat standardul POSIX. Ulterior, Linus Torvalds a scris Linux.

În 1968 a fost creată firma Intel, specializată în microprocesoare. CP/M (Control Program for Microcomputers), elaborat în 1976 de Gary Kildall, este primul sistem de operare pentru calculatoare bazate pe procesoare Intel.

1.2.4 A patra generație

După 1980, până în prezent, se poate vorbi despre a patra generație de calculatoare, reprezentate de apariția și expansiunea extraordinară a calculatoarelor personale. Producția acestora are la bază tehnologia microprocesoarelor.

Bill Gates a cumpărat sistemul de operare DOS (Disk Operating System) și l-a adaptat cerințelor IBM creând MS-DOS. Sistemul a câștigat o poziție dominantă pe piață datorită deciziei lui Gates de a-l direcționa către companiile producătoare de hardware, în special IBM.

În 1981 au apărut primele calculatoare personale IBM, costând 2880 USD, incluzând 64 KB de memorie și un floppy disk. Piața a crescut rapid, în scurt timp apărând și alte modele care foloseau aceeași arhitectură, dat fiind că IBM a decis să o facă publică. Vasta majoritate a acestor calculatoare rulau sub MS-DOS.

Doug Engelbart a realizat interfața grafică pentru utilizatori – GUI, care a fost apoi încoporată în echipamentele construite de Xerox PARC. Steve Jobs, co-inventatorul calculatorului Apple, a transformat GUI într-un succes comercial remarcabil prin lansarea Apple Macintosh în 1984 (dotat cu mouse) și direcționarea acestuia către utilizatorii cu cunoștințe sau aptitudini tehnice minimale.

Succesul lui Macintosh a inspirat Microsoft în elaborarea Windows, conceput inițial ca o interfață grafică peste MS-DOS. În 1995 a apărut prima versiune autonomă de Windows, urmând apoi Windows 98. Windows NT, o versiune complet nouă de Windows, s-a impus treptat pe piața utilizatorilor corporativi, fiind orientat spre servere și oferind posibilități avansate de administrare. A fost urmat de Windows 2000 și Windows 2003 Server. În 2001 arhitectura NT a migrat către utilizatorii individuali prin lansarea lui Windows XP. Prin Windows Vista, și actualmente Windows 7, se dorește păstrarea acestor orientări simultane atât spre partea de server cât și spre partea de utilizator.

După anii '90, sistemele Unix (în special Linux) încep să câștige teren, preponderent pe piața sistemelor server, iar apoi piața sistemelor desktop. La sfârșitul anilor '80, la MIT a apărut o primă interfață grafică pentru sistemele Unix – X Windows, urmată de versiuni mai complexe.

1.3 Sisteme de operare moderne

Deși există mai multe criterii de clasificare, doar anumite tipuri de sisteme de operare vor avea relevanță pentru utilizator obișnuit al unui calculator, fie el și unul cu profil tehnic. Un criteriu pragmatic, care ține cont de răspândirea sistemelor de operare actuale, conduce la clasificarea sistemelor de operare în:

- **sisteme de operare desktop**, destinate sistemelor de calcul obișnuite (desktop) și laptop-urilor;
- **sisteme de operare server**, destinate sistemelor de calcul folosite în general în Internet pentru a oferi servicii;
- **sisteme de operare pentru dispozitive mobile**, destinate telefoanelor mobile, smartphone-urilor sau dispozitivelor de tip PDA sau Palm.

1.3.1 Sisteme de operare desktop și server

Piața sistemelor de operare desktop este dominată în ultimele decenii de sistemele de operare din familia Windows[1]. În prezent, familia Windows deține o cotă de circa 93% din piața sistemelor desktop. Mac OS X deține circa 4.5%-5%, iar Linux în jur de 1%. Toate aceste sisteme de operare rulează în principal pe arhitecturi PC (x86 sau x86_64). Deși cu o cotă pe piață foarte mică raportat cu familia Windows, Mac OS X și Linux au câștigat procente importante pe piață.

Ținând cont că un utilizator poate opta între mai multe sisteme de operare, apar întrebări firești legate de asemănările și deosebirile dintre ele, care sunt punctele tari și punctele slabe ale fiecăruia și, în ultimă instanță, care dintre ele este cel mai bun. În realitate însă, a compara aceste cele trei clase de sisteme este un lucru foarte dificil, date fiind diferențele fundamentale de concept și realizare, iar mai mult, a trage concluzia că unul dintre sisteme este implicit mai bun sau mai slab decat celălalt este imposibil.

Toate cele trei familii de sisteme de operare oferă performanțe avansate, un mediu stabil de operare și au un domeniu foarte bogat de aplicații disponibile și concepute special pentru acestea. Dar pentru că filozofiile din spatele manierei în care sistemele au fost proiectate și au evoluat sunt diferite, concluzia finală depinde în mare măsură de scopul în care sistemul va fi folosit, gradul de pregătire al utilizatorilor, configurația hardware și mulți alți factori subiectivi.

Sistemele de operare din familia Windows și Unix/Linux oferă variante atât pentru sisteme desktop cât și pentru sisteme server. Astfel, Windows XP, Windows Vista, Windows 7 sunt versiuni desktop ale sistemului de operare de la Microsoft, în vreme ce Windows Server 2003, Windows Server 2008 sunt versiuni dedicate pentru sisteme server. În lunea Unix/Linux diferența este mai puțin clară, un sistem putând fi configurat pentru a funcționa atât ca server cât și ca desktop. Anumite distribuții Linux sunt, însă dedicate pentru sisteme desktop (Ubuntu, Fedora, openSUSE, Mint, PCLinuxOS) sau pentru sisteme server (Ubuntu Server, RedHat Enterprise Linux, SUSE Linux Enterprise). Alte sisteme de operare din familia Unix (OpenSolaris, FreeBSD, NetBSD, OpenBSD) pot fi folosite și ca sistem desktop și ca sistem server. Apple oferă Mac OS X ca sistem desktop și Mac OS X Server ca sistem server.

Windows vs. Linux

Un subiect care stârnește mult interes și dezbateri este diferența dintre Windows și Linux. De multe ori această diferență este una de ordin semantic.

Microsoft Windows desemnează un produs bine delimitat, având asociate niște specificații, pe baza cărora există un preț pe piață.

De cealaltă parte, Linux nu reprezintă un produs bine definit, termenul având mai multe înțelesuri. Tehnic vorbind, Linux este numele nucleului (kernel-ului) sistemului de operare. Ca fapt divers, Linux este o marcă înregistrată a lui Linus Torvalds, inițiatorului proiectului nucleul Linux. În prezent, proiectul este încă sub coordonarea lui Linus Torvalds. Nucleul nu depinde de restul sistemului, așa cum nici restul pachetelor care compun sistemul nu depind în mod necesar de kernel, cu unele excepții notabile.

[1] http://en.wikipedia.org/wiki/Usage_share_of_desktop_operating_systems

Cu toate acestea, majoritatea sistemelor au ajuns să fie construite pe serie comună de pachete, bazată pe proiectul GNU[1], întreg sistemul fiind pus laolaltă în ceea ce se cheamă o distribuție: o colecție de pachete, împreună cu nucleul Linux, de întreținerea și actualizarea careia se ocupă o anumită organizație sau comunitate.

Astfel, printr-un abuz de limbaj unanim acceptat, aceste clase de sisteme de operare sunt denumite distribuții Linux (sau GNU/Linux), dată fiind legătura puternică cu nucleul și pachetele de bază construite peste acesta. Întrucât Linux este numele nucleului sistemului de operare, recomandăm folosirea sintagmei *nucleul Linux* sau *kernel-ul Linux* în loc de *nucleul de Linux* sau *kernel-ul de Linux*.

1.3.2 Familia Windows

Microsoft a lansat la început Windows ca pe o colecție de aplicații MS-DOS, rulate într-un mediu grafic interactiv, care să ușureze munca utilizatorului de a lucra cu sistemul. Primele versiuni folosite pe scară largă au fost Windows 3.0 și Windows 3.1.

Un punct important l-a constituit lansarea Windows 95 în 1995. Cu o interfața intuitivă și prietenoasă, suita bogată de aplicații preinstalate, suportul tehnic dedicat, Microsoft a promovat Windows 95 și edițiile ulterioare ca fiind alegerea ideală pentru sistemul desktop de uz personal sau pentru birou.

Acest atu a fost păstrat și la edițiile ulterioare de Windows, unde s-au făcut progrese semnificative în direcția performanțelor: sistemul s-a desprins de MS-DOS, capătând un kernel modern, cu suport de multitasking și mai mulți utilizatori, interfața grafică a devenit mai bogată și mai atrăgătoare și, nu în ultimul rând, necesitatea de cunoștinte tehnice din partea utilizatorilor s-a diminuat de la ediție la ediție.

Edițiile actuale de Windows sunt construite pe tehnologia NT. Din familia NT fac parte Windows NT, Windows 2000, Windows XP, Windows Vista, Windows 7 și versiunile server lansate de Microsoft. Tehnologia NT a însemnat rescrierea nucleului sistemului de operare și obținerea unui sistem de operare modern cu funcționalități de integritate și securitate bine definite.

În momentul scrierii acestei cărți, ultima versiune desktop este Windows Vista, iar ultima versiune server este Windows Server 2008. Pe data de 22 octombrie 2009 este planificată lansarea Windows 7.

Cu o cotă impresionantă pe piața sistemelor de operare, Windows are rolul unui sistem de operare universal. Cea mai mare parte a aplicațiilor existente rulează pe sistemele Windows. Cota mare de piață a făcut Windows ținta a numeroase atacuri de securitate, rezultând în creșterea eforturilor Microsoft pentru asigurarea integrității sistemului la ultimele versiuni de Windows.

1.3.3 Mac OS X

Mac OS X reprezintă o familie de sisteme de operare concepute special pentru calculatoarele Macintosh ("Mac"-uri) produse de firma Apple. Introduse inițial în 1984,

[1] http://www.gnu.org/

sistemele de operare pentru Macintosh au cunoscut o evoluție spectaculoasă până în prezent, menținându-se drept concurent pentru Microsoft Windows atât prin performanțele sistemului propriu-zis cât și prin versatilitatea și utilitatea aplicațiilor pe care le dețin.

Calculatoarele Macintosh au implementat, de la lansare, o arhitectură hardware de tip Power PC, migrând mai apoi către o arhitectură Intel, moment în care sistemele de operare pentru Mac-uri au început să reprezinte o concurență pentru Windows.

Sistemul de operare a fost radical schimbat la lansarea versiunii 10.0, prin rescrierea nucleului. Această schimbare este reflectată și în modificarea denumirii sistemului de operare din Mac OS în Mac OS X. Adăugarea literei X subliniază trecerea de la tehnologia Mac OS nanokernel, la tehnologie derivată din Unix.

În prezent, varianta curentă de Mac OS X este 10.6, lansată la data de 28 august 2009.

Prin sistemul de operare și prin aplicațiile care pot rula pe acesta, Mac OS X excelează la capitolul interfață grafică și în domeniul prelucrării media, aducând în același timp inovații orientate spre nevoile utilizatorului individual.

1.3.4 Linux

Una dintre principalele confuzii pentru un utilizator proaspăt intrat în lumea sistemelor de operare este cea provocată de denumirile Linux și Unix. În general, Unix reprezintă o familie de sisteme de operare. Un sistem de operare este Unix (adică face parte din familia Unix) dacă respectă standardul *Single Unix Specification*[1]. În afară de Linux (sau mai bine spus GNU/Linux), alte Unix-uri (adică sisteme compatibile cu Single Unix Specification) sunt Solaris, sistemele de familia BSD (FreeBSD, OpenBSD, NetBSD) și Mac OS X.

Sistemul de operare Unix a apărut la începutul anilor '70, în laboratoarele Bell ale AT&T. Folosirea limbajului C pentru scrierea sistemului de operare a condus la portarea rapidă a acestuia și răspândirea sa în mediul acadamic. Versiunile System V (de la AT&T) și BSD 4.3 au marcat o serie de avansuri importante în anii '80.

Linux este de fapt o clonă de Unix. O clonă Unix este un sistem de operare care respectă Single Unix Specification, dar care nu conține componente din sursele inițialte Unix. BSD-urile sau Solaris sunt direct derivate din codul Unix și pot fi denumite Unix-uri veritabile.

Linux a fost scris de la zero de Linus Torvalds în 1991. Treptat, în jurul său s-a consolidat o comunitate entuziastă de dezvoltatori, Linux ajungând în acest moment unul dintre cele mai mari proiecte open-source. Dincolo de funcționalitățile oferite, popularitatea Linux în ultimii 15 ani s-a datorat apariției acestuia într-un context favorabil:

- dezvoltatorii proiectului GNU nu reușiseră crearea unui kernel;
- cei care urmăreau dezvoltarea viitoarelor sisteme BSD sub o licență liberă au pierdut câțiva ani în procese cu cei de la AT&T legate de licențierea codului.

Linus Torvalds a spus că dacă 386BSD ar fi fost disponibil la acea vreme, probabil că nu ar fi creat Linux.

[1] http://en.wikipedia.org/wiki/Single_UNIX_Specification

După apariția nucleului Linux, o multitudine de proiecte software având la bază acest kernel au fost înființate, cu numărul și importanța acestora crescând de la an la an. Spre deosebire de cazul Windows-ului, software-ul pe Linux nu a fost scris neapărat având în minte ușurința de utilizare și accesibilitatea, deși aceasta este tendința din prezent.

Un nucleu este doar un mediator între aplicații și resursele hardware – este inutil de unul singur. Pentru a obține un sistem de operare complet, diverse comunități sau companii au combinat utilitarele GNU și nucleul Linux. Au apărut astfel distribuțiile GNU/Linux (Ubuntu, Fedora, Gentoo, Debian, Slackware, OpenSuSE, RedHat etc.), care oferă utilizatorului funcționalități complete pentru operarea sistemului. Din punct de vedere strict tehnic, expresia corectă este *instalarea unei distribuții Linux (GNU/Linux)*, nu *instalarea Linux*. Ținând cont de prevalența formei *distribuție Linux*, nu vom insista asupra acestui lucru pe parcursul acesi carți pe justețea exprimării (vezi secțiunea 2.1).

Fiind vorba îndeosebi de mediul academic, majoritatea utilizatorilor aveau cunoștințe avansate de IT și apreciau la un sistem stabilitatea, flexibilitatea și performanța, mai degrabă decât o interfață simplă și intuitivă. Datorită acestor puternice avantaje, Linux (ca majoritatea sistemelor Unix, de altfel) a avut un succes exploziv pe servere, și a câștigat teren într-o măsură mult mai mică pe desktop, unde supremația este încă deținută de către sistemele de operare de la Microsoft. În momentul de față, o bună parte din infrastructura Internet-ului se bazează pe sisteme rulând Linux.

1.3.5 Alte Unix-uri

Având la bază un kernel Mach[1] și facilități BSD, **Mac OS X** poate fi considerat un sistem Unix-like. De la Mac OS X 10.5 Leopard, sistemul de operare de la Apple respectă *Single Unix Specification* și este un Unix veritabil[2].

Solaris[3] este sistemul de operare dezvoltat de Sun Microsystems. Inițial dezvoltat în format proprietar (*closed-source*), majoritatea componentelor sunt acum open-source. **OpenSolaris**[4] este o distribuție open-source a sistemului de operare de la Sun, utilizabil și în forma unui LiveCD.

Distribuțiile **BSD** (*Berkeley Software Distribution*) sunt distribuțiile derivate din codul 4.4BSD-Lite, ultima versiune de Unix de la Berkeley. Aceste distribuții folosesc o licență liberă (licență BSD). Licența fiind permisivă, o parte din componentele software sunt integrate în Windows, Mac OS X și Solaris. Distribuțiile BSD sunt asociate unei distribuții GNU/Linux: nu conțin doar nucleul, ci și toate aplicațiile necesare folosirii sistemului. Dezvoltarea întregii distribuții se realizează centralizat. Distribuțiile BSD cele mai folosite astăzi sunt:

- **FreeBSD** – cel mai folosit sistem de operare derivat din BSD – axat pe oferirea unui sistem de operare stabil și utilizabil;
- **OpenBSD** – sistem de operare derivat din NetBSD, puternic axat pe securitate;

[1] http://en.wikipedia.org/wiki/Mach_kernel
[2] http://arstechnica.com/apple/news/2007/08/mac-os-x-leopard-receives-unix-03-certification.ars
[3] http://www.sun.com/software/solaris/
[4] http://opensolaris.org/os/

- **NetBSD** – recunoscut pentru portabilitatea sa; este folosit ca punct de plecare pentru portarea altor sisteme de operare pe noi arhitecturi.

1.3.6 Comunități open source

Principala diferență dintre Linux și multe alte sisteme de operare actuale, mai mult sau mai puțin răspândite, este faptul că atât nucleul Linux cât și componentele unui sistem GNU/Linux sunt gratuite. Mai mult decât atât, sunt free/open-source. Acest lucru înseamnă că o comunitate vastă de utilizatori și ingineri software au acces la ele, pentru a le utiliza sau a le aduce noi îmbunătățiri. Acestă practică nu este nouă, ci datează de la începuturile Unix la MIT labs, când echipele inițiale de hackeri lucrau în comun pentru dezvoltarea de aplicații Unix.

Conceptul de licență sub care funcționează Linux, alături de aplicațiile sale, a primit denumirea de *copyleft* care, contrar copyright-ului, oferă oricui dreptul de acces, distribuție și modificare a unei creații, atâta timp cât păstrează aceleași caracteristici ale *copyleft-ului*. La polul opus, majoritatea sistemelor de operare ce nu aparțin familiei Linux și, cu preponderență cele comerciale, se supun în continuare legii copyright-ului și împiedică intervenția utilizatorilor asupra codului sursă (closed source).

Comunitățile open source sunt foarte dinamice, iar, în acest moment, distribuțiile de Linux beneficiază din plin de acest lucru. Până acum câțiva ani, sistemele Linux necesitau cunoștințe avansate de operare, din motive legate de instrumente grafice de administrare limitate sau lipsa driverelor din partea producătorilor de hardware. Mai mult, accesul limitat la Internet priva utilizatorul obișnuit de accesul la documentație (deși vastă, în mare parte online), esențială pentru încurajarea utilizatorilor de a experimenta cu un nou sistem de operare, în contextul în care nu existau campanii de marketing ca în cazul produselor comerciale.

Lucrurile au început totuși să se schimbe într-un mod radical în momentul în care Internet-ul a devenit o resursă din ce în ce mai accesibilă și o serie de companii au remarcat potențialul pe care Linux îl avea și și-au orientat domeniul de activitate către acesta. Interesul crescut a încurajat comunitățile să depună un efort mai mare pentru a face aplicațiile mai accesibile.

Robustețea și performanțele superioare au fost îmbinate cu o interfață bogată și o experiență vizuală comparabilă (dacă nu superioară – vezi proiectele Compiz și Beryl) cu cea de pe Windows. Suportul din partea producătorilor de hardware a crescut la rândul său, cu implicări foarte puternice din partea unor nume ca Novell, Canonical, sau Dell. Astfel, desktop-ul Linux devine o opțiune atrăgătoare, anii următori anunțându-se extrem de promițători pentru ascensiunea mișcării Open Source în mediul desktop.

Există, totuși, reticențe cu privire la încurajarea răspândirii programelor open-source din moment ce o astfel de libertate a tuturor utilizatorilor este dificil de controlat și poate crea probleme de interoperabilitate. Într-adevăr, unificarea pe care o oferă sistemele de operare comerciale, concepute global și, totodată, compact, impune și uniformizarea aplicațiilor scrise pentru a rula pe ele, în timp ce varietatea deja mare și în continuă creștere a sistemelor open-source conduce la stabilirea unei multitudini de direcții de dezvoltare între care se împarte software-ul conceput pentru toate acestea.

Pe de altă parte, comunitatea de utilizatori ai programelor open-source este în continuă

creștere iar dezvoltarea intensă a distribuțiilor Linux conduce la crearea de pachete specifice care facilitează accesul la aplicațiile necesare. În momentul de față, versiunea stabilă a distribuției Debian GNU/Linux oferă peste 25 de mii de pachete[1].

Cea mai mare comunitate de programatori ce dezvoltă aplicații open-source pentru un număr foarte mare de platforme și sistemele de operare este SourceForge[2]. Microsoft a încurajat dezvoltarea unei comunități, numită CodePlex[3], axată pe dezvoltarea aplicațiilor open-source folosind tehnologiile Microsoft, precum .NET Framework, Windows Presentation Foundation, XNA etc. pentru sisteme de operare de la Microsoft, cu preponderență Window XP și Windows Vista.

1.3.7 Sisteme de operare pentru dispozitive mobile

O dată cu dezvoltarea hardware-ului și a Internet-ului, un număr din ce în ce mai mare de dispozitive mobile inteligente și-au făcut apariția în posesia utilizatorilor. Așa numitele *smartphone-uri*[4] posedă sisteme de operare cu facilități avansate de conectare la Internet, navigare pe Web, citire/scriere de e-mail-uri, folosirea unei tastaturi virtuale etc. Exemple de astfel de dispozitive sunt iPhone[5] de la Apple, BlackBerry[6], Nokia E71[7], HTC[8] etc.

Cel mai răspândit sistem de operare folosit pe dispozitive mobile în general, și pe smartphone-uri în particular este **Symbian OS** de la Symbian Ltd. Acesta deține aproape 50% din piața sistemelor de operare pentru smartphone-uri. Sistemul conține kernel-ul și o serie de aplicații utile în lucrul cu telefonul mobil. Symbian OS oferă o interfață de dezvoltare (SDK) în C++ pentru scrierea de noi aplicații. Înființată în 2008, Fundația Symbian are ca scop folosirea unei licențe libere pentru întreg codul platformei Symbian.

Windows Mobile este sistemul de opearare și suita de aplicații pentru dispozitive mobile de la Microsoft. Este proiectat să arate simiar interfeței Windows și este folosit cu preponderență în cadrul dispozitivelor Pocket PC. Cota de piață a Windows Mobile a scăzut constant în ultimii ani până la o valoare circa 9%.

iPhone OS este sistemul de operare dezvoltate de Apple pentru iPhone și iPod. Interfața acestuia folosește gesturi de tip multi-touch – se atinge ecranul pentru acționarea unui eveniment. iPhone OS oferă un număr impresionant de aplicații. Deține, în acest moment, aproape 15% din piața dispozitivelor mobile inteligente.

Android este un sistem de operare care folosește nucleul Linux. Android este dezvoltat de *Open Handset Alliance*, un consorțiu înființat de Google împreună cu mari companii producătoare de hardware și software precum HTC, Intel, Motorala, LG etc. Dezvoltarea acestuia se realizează în cadrul unei licențe libere (cu unele excepții), lucru care permite dezvoltarea relativ facilă de noi aplicații. În aceste momente, Android are o cotă de

[1] http://packages.debian.org/lenny/
[2] http://sourceforge.net
[3] http://www.codeplex.com/
[4] http://en.wikipedia.org/wiki/Smartphone
[5] http://www.apple.com/iphone/
[6] http://www.blackberry.com/
[7] http://europe.nokia.com/find-products/devices/nokia-e71
[8] http://www.htc.com/www/product/dream/overview.html

piață de doar 3%, cauzată și de apariția sa recentă. Dezvoltarea Android se realizează independent de platforma hardware pe care va rula.

Linux este folosit ca bază pentru mai multe sisteme de operare pentru dispozitive mobile. Pe lângă Android, LiMo, Maemo și Openmoko sunt exemple de sisteme de operare bazate pe Linux.

1.4 Studiu de caz

1.4.1 Virtualizare. Rularea unui SO dintr-o mașină virtuală

O **mașină virtuală** este o aplicație ce creează un mediu de execuție pentru rularea unui nou sistem de operare, folosind resursele gestionate de sistemul de operare gazdă, prin intermediul mecanismelor oferite de mașina virtuală.

O caracteristică particulară a mașinii virtuale este capacitatea acesteia de a izola procesele ce rulează sistemul de operare din interiorul său de cele ale sistemului de operare gazdă.

Sistemul de bază, în cadrul căruia (sau peste care) rulează mașina virtuală se cheamă **sistem gazdă (*host*)**. Mașina virtuală se mai numește și **sistem oaspete (*guest*)**.

Aplicația care permite rularea unui sistem de operare virtualizat se numește **hipervizor** (*hypervisor*) sau *virtual machine monitor* (VMM) sau sistem/soluție de virtualizare.

Exemple de soluții de virtualizare sunt VMware, VirtualBox, VirtualPC, Xen, KVM.

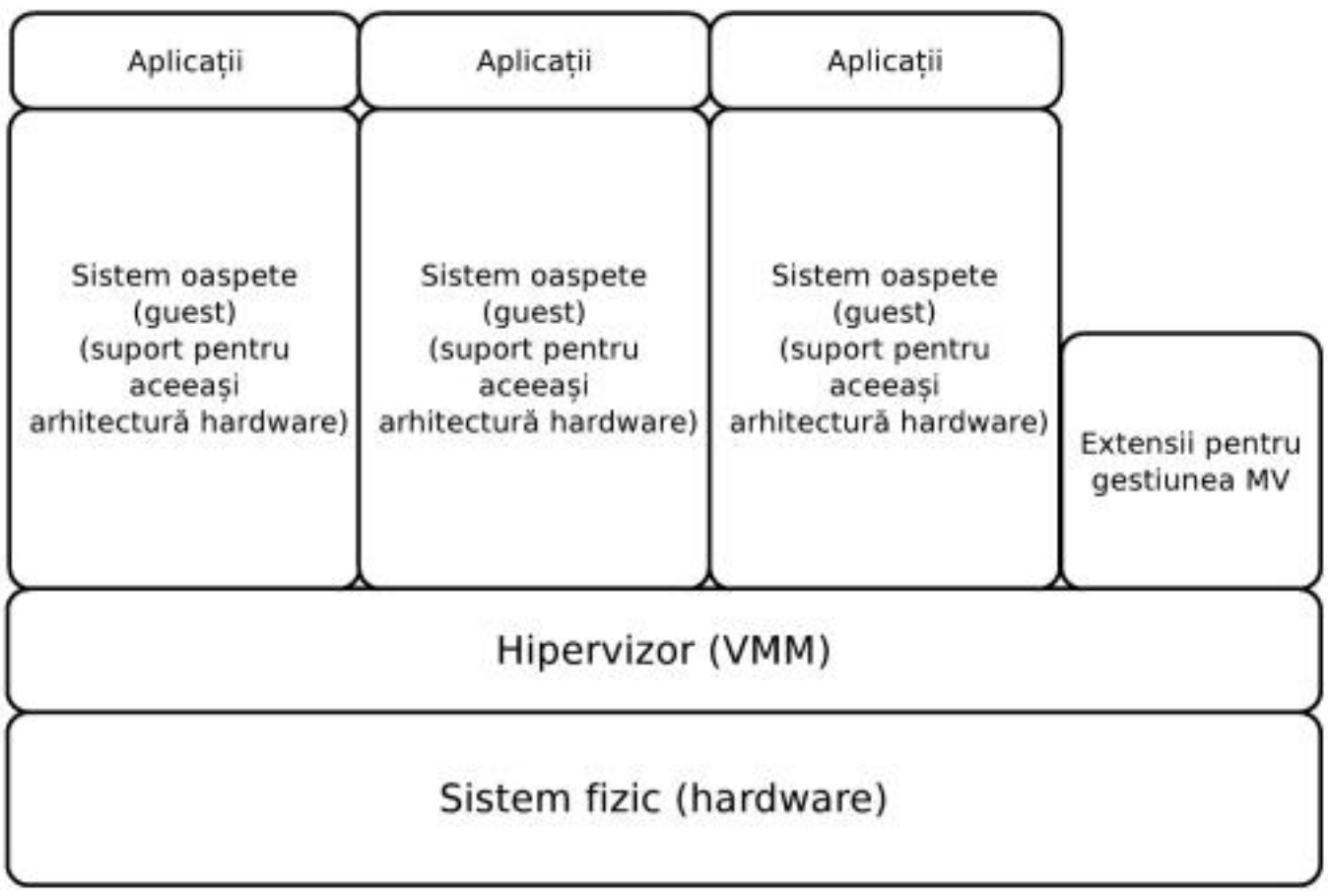


Figura 1.2: Mașină virtuală

În general, sistemul gazdă alocă părți din resursele sistemului fizic pentru mașina virtuală: un procentaj dedicat din memoria RAM, un spațiu fizic de stocare de pe hard-disk-ul calculatorului gazdă (alocat în prealabil sau în timp real) pe care mașina virtuală îl va folosi ca pe un hard-disk propriu, accesul la diferite dispozitive periferice. Accesul celui de-al doilea sistem de operare la procesor se face prin intermediul proceselor mașinii virtuale, care vor rula alături de celelalte aplicații și vor împărți aceleași resurse. Noțiunea de virtualizare nu trebuie confundată cu cea de **emulare**[1]. Un emulator traduce fiecare instrucțiune din sistemul oaspete în instrucțiuni specifice sistemului gazdă. Prin folosirea unui emulator este astfel posibilă rularea unui program pentru o arhitectură diferită de cea a sistemului gazdă. Un sistem virtualizat va folosi în proporție cât mai mare sistemul fizic gazdă, din motive de eficiență. Rolul hipervizorului este de a intermedia unele operații privilegiate care nu se pot executa direct peste sistemul fizic gazdă.

Noțiunea de virtualizare este extinsă și la nivelul aplicațiilor. Programele Java rulează în cadrul mașinii virtuale Java[2]. Mașina virtuală Java este un mediu independent de platformă care permite execuția în același context a aplicațiilor. Un alt exemplu este *Common Language Runtime*[3], mașina virtuală folosită de *.NET Framework*[4]. Valgrind (vezi secțiunea 14.8.2), folosit pentru depanarea programelor, este, de asemenea, o mașină virtuală la nivelul aplicațiilor.

De ce virtualizare?

Dezvoltate încă din anii '70, mașinile virtuale au început să fie folosite intens odată cu dezvoltarea sistemelor hardware. Creșterea capacității de calcul (procesoare multi-core, memorie) și a capacității de stocare (hard-disk-uri) a condus la apariția soluțiilor de virtualizare care să folosească eficient resursele hardware puse la dispoziție. Adăugarea de suport hardware pentru arhitecturile x86 și x86_64 (AMD-V[5] și Intel VT[6]) a condus la accelerarea folosirii soluțiilor de virtualizare atât pentru utilizatorul obișnuit cât, mai ales, în mediul business.

Unul din principalele motive pentru folosirea virtualizării, mai ales în mediul comercial, este **consolidarea încărcării sistemelor fizice**. În locul folosirii a patru sisteme fizice, se poate folosi un singur sistem fizic în cadrul căruia să ruleze patru mașini virtuale, cu efect în diminuarea costurilor de achiziție și administrare.

Un al avantaj importat în reprezintă **securizarea aplicațiilor**. Fiecare mașină virtuală este un mediu izolat și sigur (sandbox). O problemă care apare pe o mașină virtuală nu va afecta niciun alt sistem și va putea fi diagnosticată rapid folosind sistemul gazdă. Virtualizarea este un concept important pentru construirea de platforme de calcul sigure.

Soluțiile de virtualizare permit crearea de instanțe de sisteme de operare sau **medii de execuție cu limitări impuse**. Se pot configura procentaje de resurse care să fie folosite de mașina virtualizată. Se asigură astfel un control fin pentru gestiunea resurselor.

---

[1] http://en.wikipedia.org/wiki/Emulator
[2] http://www.java.com/en/download/index.jsp
[3] http://msdn.microsoft.com/en-us/library/ddk909ch(VS.71).aspx
[4] http://msdn.microsoft.com/en-us/netframework/default.aspx
[5] http://www.amd.com/us/products/technologies/virtualization/Pages/virtualization.aspx
[6] http://www.intel.com/technology/virtualization/

Pentru utilizatorul obișnuit, cel mai important avantaj al virtualizării este **posibilitatea rulării mai multor sisteme de operare**. Sisteme de operare diferite, versiune de sisteme operare diferite, pot rula simultan pe același sistem. Pornirea și repornirea unui sistem de operare oaspete se rezumă la interacțiunea cu soluția de virtualizare fără a implica acțiuni asupra sistemului fizic.

Pentru un dezvoltator, o mașină virtuală reprezintă un **mediu ideal pentru dezvoltare și testare**. Orice problemă a aplicației pe care o dezvoltă/testează va afecta doar sistemul de operare oaspete. Dezvoltarea la nivelul nucleului (kernel programming) se realizează, în general, folosind mașini virtuale. Scenarii complexe de monitorizare a performanței sau depanare pot fi implementate pe o mașină virtuală fără pierderea productivității.

Un avantaj al folosirii mașinilor virtuale este **mobilitatea acestora**. O mașină virtuală este, de obicei, văzută ca un set de fișiere. Mutarea acestora pe un alt sistem de calcul este echivalent cu migrarea mașinii virtuale pe acel sistem. Sistemul de operare din cadrul mașinii virtuale va rula în același mod pe un nou sistem. Este posibilă migrarea unei mașini virtuale în timp ce aceasta rulează, tehnică denumită *Live Migration*[1].

Exemple de mașini virtuale

O dată cu proliferarea tehnologiilor de virtualizare, ultimul deceniu a însemnat apariția mai multor companii și soluții. Printre cele mai cunoscute soluții de virtualizare actuale sunt VMware, VirtualBox, VirtualPC, Xen, KVM, OpenVZ. Aceste soluții de virtualizare rulează, în general, pe arhitecturi x86 sau x86_64.

VMware, Inc.[2] este o companie americană care produce mai multe aplicații de virtualizare. Cele mai cunoscute sunt VMware Workstation, VMware Player, VMware Server și VMware ESX. VMware Workstation, Player și Server sunt aplicații care rulează într-un sistem de operare gazdă (Windows sau Linux). VMware Fusion este versiunea pentru Mac OS X. VMware ESX, versiunea enterprise, este un sistem de operare complet peste care se instalează instanțe de mașini virtuale.

VirtualBox[3] este o soluție de virtualizare dezvoltată de Sun Microsystems. Similar cu VMware Workstation, este o aplicație care rulează peste un sistem de operare gazdă (Windows, Linux, Mac OS X, Solaris) și în cadrul cărei pot fi instalate alte sisteme de operare. VirtualBox Open Source Edition este disponibilă sub licență GNU.

Microsoft Virtual PC[4] este un sistem de virtualizare pentru sisteme de operare din familia Windows. Similar VirtualBox și VMware Workstation, Virtual PC oferă suport pentru sisteme oaspete doar din familia Windows. Se pot rula și sisteme de operare Linux, dar cu pași suplimentari de configurare.

Xen[5] este un hipervizor (virtual machine monitor) creat la Universitatea din Cambridge și dezvoltat actualmente de comunitatea Xen ca software liber, sub licență GNU. Xen este similar VMware ESX. Are suport în nucleul sistemului de operare Linux, NetBSD și Solaris. Peste sistemul de bază pot fi instalate alte sisteme de operare oaspete. Xen

[1] http://en.wikipedia.org/wiki/Live_Migration
[2] http://www.vmware.com/
[3] http://www.virtualbox.org/
[4] http://www.microsoft.com/windows/virtual-pc/
[5] http://www.xen.org/

folosește o forma de virtualizare denumită paravirtualizare care permite performanță ridică. Paravirtualizarea necesită însă modificarea sistemului de operare oaspete pentru a putea rula peste Xen. Dacă sistemul de bază oferă suport de virtualizare hardware, atunci nu mai este necesară modificarea sistemului de operare; efectul este posibilitatea rulării unui sistem Windows nemodificat peste Xen.

KVM[1] (*Kernel-based Virtual Machine*) este o infrastructură de virtualizare a nucleului Linux. KVM oferă suport pentru virtualizarea hardware (Intel VT și AMD-V). KVM este inclus în nucleul Linux începând cu versiunea 2.6.20. O parte din utilitarele necesare sunt furnizate de QEMU[2].

OpenVZ[3] este o tehnologie la nivelul sistemului de operare bazată pe Linux. Licențiat GPL, OpenVZ constituie baza pentru Parallels Virtuozzo Containers, o aplicație proprietară furnizată de Parallels, Inc.[4] OpenVZ este mai degrabă un container decât o mașină virtuală. OpenVZ nu virtualizează întreg sistemul de operare. Nucleul sistemului de operare gazdă este folosit de toate mașinile virtuale (denumite și containere sau medii virtuale). Procesele din cadrul fiecărui container sunt izolate, dar se folosește același nucleu. Prezența aceluiași nucleu înseamnă, totuși, overhead redus pentru lucrul cu diversele containere rezultând în viteză și performanță ridicate.

Cuvinte cheie

- programe de bază
- sistem de operare
- nucleu (kernel)
- portabilitate
- API
- biblioteci
- Windows
- Mac OS X
- Linux
- Unix
- open-source
- smartphone
- virtualizare
- hipervizor
- VMware
- VirtualBox
- VirtualPC
- Xen
- KVM
- OpenVZ

Întrebări

1. Care dintre următoarele aplicații poate fi privită ca o componentă a sistemului de operare?
   - ❑ procesor de text
   - ❑ player pentru fișiere video

[1] http://www.linux-kvm.org/page/Main_Page
[2] http://www.nongnu.org/qemu/
[3] http://wiki.openvz.org/Main_Page
[4] http://www.parallels.com/eu/

- ❑ interpretor de comenzi
- ❑ client de email

2. Care dintre următoarele NU reprezintă o resursă partajată în cazul virtualizării?
   - ❑ RAM
   - ❑ hard-disk
   - ❑ procesor
   - ❑ BIOS
3. Care dintre următorii este autorul unui limbaj de programare?
   - ❑ Larry Wall
   - ❑ Linus Torvalds
   - ❑ William Henry Gates III
   - ❑ Andrew S. Tanenbaum
4. Care dintre următoarele NU este o funcție a sistemului de operare?
   - ❑ controlul accesului la memorie
   - ❑ căutarea și eliminarea programelor virus
   - ❑ asigurarea comunicării între procese
   - ❑ organizarea fișierelor în directoare
5. Care din următoarele sisteme de operare NU este folosit pentru dispozitive mobile?
   - ❑ OpenSolaris
   - ❑ iPhone OS
   - ❑ Symbian OS
   - ❑ Android
6. Care din următoarele tehnologii de virtualizare rulează NUMAI pe un sistem Linux?
   - ❑ VirtualBox
   - ❑ VMware
   - ❑ VirtualPC
   - ❑ OpenVZ
7. Ce reprezintă termenul GNU din *distribuție GNU/Linux*?
   - ❑ o antilopă din Africa
   - ❑ o anagramă a cuvântului *gun* (armă)
   - ❑ un proiect care produce o componentă importantă a aplicațiilor ce rulează peste nucleul Linux
   - ❑ compania care se ocupă de dezvoltarea nucleului Linux

8. Care din următoarele sisteme de operare are cea mai mare cotă pe piața sistemelor desktop?
   - ❑ Windows
   - ❑ Mac OS X
   - ❑ Linux
   - ❑ OpenSolaris
9. Care din următoarele sisteme de operare are cea mai mare cotă pe piața dispozitivelor mobile inteligente?
   - ❑ Windows Mobile
   - ❑ iPhone OS
   - ❑ Android
   - ❑ Symbian OS
10. Care din următoarele sisteme de operare este cel mai vechi?
   - ❑ DOS
   - ❑ Unix
   - ❑ Linux
   - ❑ OpenBSD

Capitolul 2

Instalarea Linux. Configurări de bază

Unix is the answer, but only if you phrase the question very carefully.

Ce se învață din acest capitol?

- Ce este Linux
- Distribuții GNU/Linux
- Partiționarea discului; pregătirea pentru instalare
- Instalarea Kubuntu
- Interfața cu utilizatorul; interfața în linia de comandă
- Autentificarea în sistem
- Configurări de bază într-un sistem Linux
- Interoperabilitate Linux/Windows

2.1 Linux. Distribuții Linux

Termenul "Linux" este deseori folosit pentru a reprezenta o întreagă clasă de sisteme de operare și nu un sistem de operare concret. Inițial, termenul "Linux" a fost folosit pentru a referi nucleul Linux[1] (*Linux kernel*), sistemele de operare bazate pe Linux și pe utilitarele din proiectul GNU[2] fiind numite *sisteme de operare GNU/Linux*. Ulterior, termenul a ajuns să fie folosit pentru a referi atât nucleul Linux cât și programele care funcționează cu acesta.

O mare parte din utilitarele folosite în Linux sunt dezvoltate ca proiecte open source (vezi capitolul 14) și, deși sunt dezvoltate într-o manieră colaborativă, sunt produse independent unele de altele. Licențele sub care se afla aceste proiecte permit

[1] http://www.kernel.org/
[2] http://www.gnu.org/

redistribuirea codului sursă, făcând astfel posibilă apariția unor noi proiecte mai mari, care să adune la un loc programe dezvoltate independent și să le facă disponibile la un loc sub forma unei distribuții.

> O **distribuție Linux** reprezintă astfel o colecție de programe, unele din ele modificate special pentru acea distribuție, care sunt combinate cu o versiune de kernel, de multe ori și aceasta modificată pentru acea distribuție, pentru a crea un sistem de operare.

Distribuțiile Linux au căpătat de-a lungul timpului cele mai variate forme și destinații. Unele au ca scop principal utilizarea lor pe servere, altele sunt destinate utilizării pe Ssisteme desktop, iar altele sunt optimizate pentru a ocupa cât mai puține resurse putând astfel fi folosite în sistemele embedded.

2.1.1 Principalele distribuții Linux

În momentul de față există peste 300 de distribuții active (care sunt dezvoltate în continuare). Dintre ele doar puțin peste 10 sunt cunoscute la scară largă, fiind utilizate pentru scopuri generale.

De cele mai multe ori o distribuție nu este dezvoltată de la zero, punctul de plecare constituindu-l un anumit stadiu al unei alte distribuții. Noua distribuție poate moșteni de la cea din care este dezvoltată diverse elemente, cum ar fi modul în care se instalează aplicații noi, locul în care se găsesc diverse fișiere de configurare, felul în care arată aceste fișiere etc.

Popularitatea distribuțiilor poate fi urmărită pe pagina web http://distrowatch.com. Aici este realizat un clasament al distribuțiilor pe baza numărului de accesări zilnice ale informațiilor despre distribuție pe pagina amintită. Acest clasament nu ia în considerare efortul depus pentru menținerea distribuției, dar arată în mare măsură interesul manifestat de utilizatori.

Cele mai accesate 16 distribuții în ultimul an (la data de 19.09.2009) sunt prezentate în tabelul 2.1 (coloana "Accese" prezintă numărul mediu de accese zilnice pe perioada ultimul an; Distribuția *SLS* este descrisă online[1]).

Pe lângă distribuțiile de uz general (pentru servere sau pentru calculatoare personale) există și distribuții dedicate unui scop precis (*one purpuse*) – spre exemplu **Musix**[2], distribuție dedicată producerii de material multimedia, **SystemRescueCD**[3], distribuție Live CD (vezi secțiunea 2.2.3) specializată în repararea unui calculator pe care sistemul de operare nu se mai inițializează din motive ce țin de software, sau **BackTrack**[4], distribuție Live CD care oferă utilitarele necesare atacării unei rețele de calculatoare.

O clasă aparte de distribuții o reprezintă cele care au ca țintă ocuparea a cât mai puțin spațiu. Cea mai cunoscută distribuție ce urmărește acest lucru este **DamnSmallLinux**[5], distribuție care reușește să includă în doar 50MB o distribuție completă cu interfață grafică și cu toate aplicațiile uzuale.

---

[1] http://en.wikipedia.org/wiki/Softlanding_Linux_System
[2] http://www.musix.org.ar/en/
[3] http://www.sysresccd.org/
[4] http://www.remote-exploit.org/backtrack.html
[5] http://www.damnsmalllinux.org/

Tabelul 2.1: Cele mai accesate distribuții pe distrowatch.com (ultimul an)

Poz.	Nume	Anul apariției	Derivată din	Accese
1	Ubuntu	2004	Debian	2221<
2	openSUSE	1994	Slackware	1454=
3	Fedora	2003	Red Hat	1441>
4	Mint	2006	Ubuntu	1401<
5	Debian	1993	–	1074<
6	Mandriva	1998	Red Hat	970=
7	PCLinuxOS	2003	Mandriva (Mandrake)	853<
8	Puppy	2005	–	718>
9	Sabayon	2006	Gentoo	667>
10	CentOS	2003	Red Hat	626<
11	Arch	2002	–	617>
12	MEPIS	2003	Debian	586=
13	Slackware	1993	SLS	561>
14	FreeBSD	1993	386BSD	491>
15	Kubuntu	2005	Ubuntu	427<
16	Damn Small	2003	Knoppix	423<

Pe lângă acestea, comunitățile au dezvoltat distribuții caracteristice unor anumite zone geografice – **Kiwi Linux**[1] este o distribuție adaptată utilizatorilor din România.

Pe Internet sunt disponibile mai multe diagrame care prezintă evoluția distribuțiilor și felul în care au derivat una din alta. O simplă căutare a șirului de caractere "*linux distro timeline*" pe *Google Images*[2] va oferi câteva zeci de variante ale acestei diagrame. O versiune care conține doar principalele distribuții este prezentată în figura 2.1.

2.1.2 Debian. Ubuntu

Una dintre primele distribuții apărute este **Debian**. Din 1993 și până în prezent această distribuție a reprezentat punctul de plecare a peste 40 de distribuții.

Unul dintre cele mai mari avantaje ale distribuției îl reprezintă suportul bun disponibil pe Internet, atât pe pagina web oficială[3] cât și pe alte site-uri. Distribuția oferă posibilitatea de a instala și configura sistemul de operare folosind pachete, fiind disponibile peste 25000 de pachete[4] – există disponibil suport pentru actualizarea facilă și organizată a sistemului de operare. De asemenea sunt disponibile trei variante ale distribuției: *stable*, *testing* și *unstable*, exitând astfel posibilitatea alegerii gradului de stabilitate și noutate dorit.

Cea mai cunoscută distribuție derivată din Debian este **Ubuntu**[5]. În momentul de față Ubuntu este și distribuția cu cea mai mare popularitate dintre toate distribuțiile existente, fiind pe primul loc în clasamentul distrowatch.com de aproape patru ani. Atuul principal

[1] http://kiwilinux.org/
[2] http://images.google.com/
[3] http://www.debian.org/support/
[4] în versiunea Debian 5.0 (*lenny*)
[5] http://www.ubuntu.com

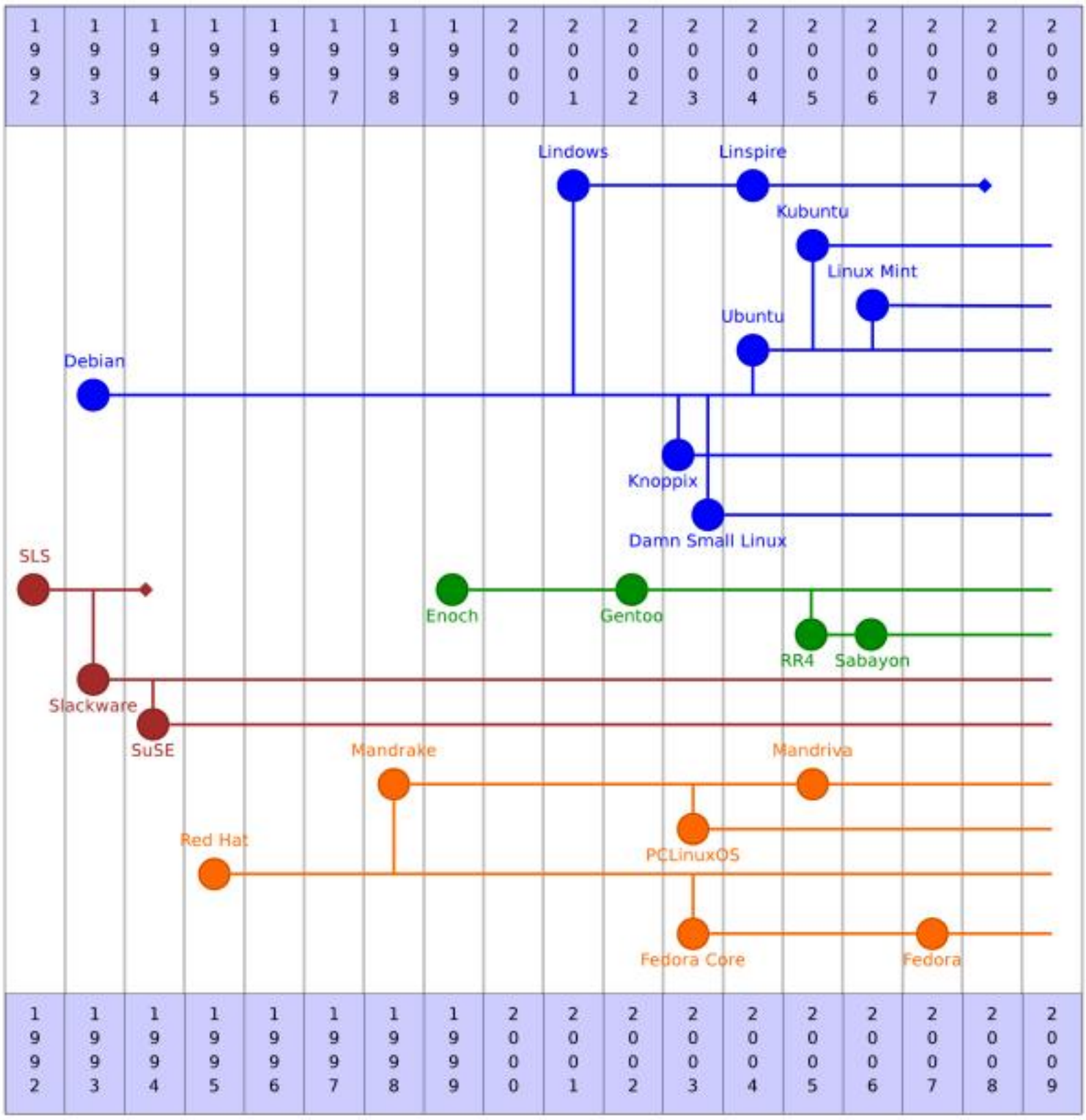


Figura 2.1: Evoluția principalelor distribuții Linux

al acestei distribuții este ușurința în utilizare. Distribuția conține toate aplicațiile de care un utilizator are nevoie și realizează multe dintre configurări în mod automat. Un avantaj mare pe care îl oferă este suportul disponibil pe forumul oficial.

Nivelul de adopție al acestui sistem de operare a crescut ușor odată cu distribuirea acestuia preinstalat pe calculatoare. Prima inițiativă în acest sens a avut-o unul dintre cel mai mari producători de echipamente IT care, în mai 2007, a anunțat că va scoate pe piață calculatoare desktop și laptop-uri care au preinstalat Ubuntu. În prezent, un mare număr de computere desktop, laptop dar și de tip netbook sunt vândute cu Linux preinstalat.

2.1.3 Tipuri de distribuții Linux

În IT există două concepte importante legate de stabilitatea, siguranța și gradul de noutate al aplicațiilor (și nu numai): versiunile stabile (*stable*) și cele cu un grad de noutate ridicat (*bleeding edge*). Aceste două concepte reprezintă două extreme ale modului în care este dezvoltată o aplicație.

Versiunile stabile (*stable*) reprezintă versiuni ale programelor care au fost testate un timp îndelungat, ale căror bug-uri au fost rezolvate, care nu prezintă probleme de incompatibilitate și care au un grad de siguranță ridicat. Într-o versiune stabilă este destul de improbabil să apară breșe de securitate. Aceste versiuni apar rar deoarece este nevoie de timp pentru a crea stabilitatea necesară.

Dezavantajul versiunilor stabile este acela că aplicațiile nu prezintă niciun grad de noutate. De multe ori pot să dureze ani până când o nouă versiune a unei aplicații să fie introdusă într-o versiune stabilă a unei distribuții Linux.

Versiunile stabile sunt folosite în special pentru severe datorită gradului de stabilitate și siguranță oferit.

Versiunile cu un grad de noutate ridicat (*bleeding edge*) reprezintă versiuni recente ale programelor, versiuni care includ ultimele îmbunătățiri și inovații. Aceste versiuni apar destul de des, frecvența lor de apariție fiind în mare măsură influențată de mărimea comunității ce dezvoltă aplicația și de îmbunătățirile aduse.

Dezavantajul versiunilor cu grad de noutate ridicat este acela că prezintă un grad de stabilitate și de siguranță incert, de multe ori scăzut. Datorită testării slabe a aplicațiilor (din lipsă de timp), acestea prezintă deseori găuri de securitate sau probleme de stabilitate.

Versiunile cu grad de noutate ridicat sunt recomandate celor care doresc să aibă cele mai noi tehnologii implementate cu orice cost. Ele nu sunt recomandate pentru servere, ci pentru calculatoare unde stabilitatea și siguranța nu sunt elemente cheie.

Între cele două tipuri de aplicați prezentate există numeroase niveluri de compromis. Alegerea unui astfel de nivel pentru un sistem de operare (mai aproape de stabilitate sau mai aproape de cele mai noi tehnologii) trebuie făcută ținând cont de scopul calculatorului pe care va fi instalat și de gradul de securitate și de stabilitate pe care trebuie să îl prezinte acesta.

Distribuțiile au moduri diferite de tratare a conceptelor *stable* și *bleeding-edge*.

Spre exemplu, distribuția Debian poate fi instalată în 3 versiuni:

- *stable* – versiune stabilă, cu aplicații testate mult timp
- *testing* – versiune de testare, după o perioadă de timp devine versiunea *stable*
- *unstable* – versiune care incorporează programele în versiunile lor *bleeding-edge*, devine versiunea testing după o perioadă

Ubuntu are un ciclu de dezvoltare de 6 luni (în engleză *release cycle* – ciclu de lansare). Acest lucru presupune dezvoltarea distribuției timp de 6 luni și lansarea unei noi versiuni după această perioadă. În prezent, Ubuntu lansează distribuții în aprilie (numărul X.04) și în octombrie (numărul X.10) – X reprezintă ultimele două cifre din an (pentru 2009 – 9, 2010 – 10 etc.).

Pe lângă aceste versiuni, Ubuntu are și o versiune LTS (*Long Term Support*) cu un ciclu de dezvoltare de 2 ani, versiune pentru care este oferit suport timp de 3 ani pentru sisteme desktop (5 ani pentru sisteme server).

Pentru fiecare versiune de Ubuntu/Kubuntu, cele mai utilizate trei variante de instalare sunt:

- **Instalarea server** este mai apropiată de Debian și de stabilitatea acestuia. Este recomandată pentru calculatoare care trebuie să ofere stabilitate și siguranță.
- **Instalarea desktop** (cea mai uzuală) permite utilizatorului să instaleze varianta desktop a sistemului de operare folosind un asistent în mod grafic. Pentru o bună funcționare, se recomandă utilizarea unui sistem cu cel putin 384MB de RAM.
- **Instalarea alternate** permite utilizatorului să instaleze varianta desktop, însă asistentul de instalare nu va mai fi unul în mod grafic ci unul în mod text, bazat pe asistentul de instalare al distribuției Debian. Din acest motiv varianta alternate poate fi instalată și pe sisteme cu mai puțin de 384 MB de RAM. Un alt avantaj al variantei alternate este acela că permitea realizarea unei partiționări avansate a hard-disk-ului, oferind suport pentru utilizarea RAID[1] sau LVM[2] (ambele tehnologii permit realizarea de partiții care includ spațiul de pe mai multe hard-disk-uri).

Pe lângă aceste versiune mai există și versiunea *Netbook Remix* (pentru calculatoare de tip *netbook*).

2.2 Instalarea Linux

2.2.1 Alegerea distribuției Linux

După cum a fost menționat și în secțiunea 2.1, există mai multe tipuri de distribuții. Câteva criterii de alegere a unei distribuții au fost menționate anterior.

Principalele criterii folosite în alegerea distribuțiilor sunt:

- nivelul de stabilitate și gradul de noutate al aplicațiilor (în relație cu nivelul de stabilitate, după cum a fost menționat anterior)
- suportul oferit de comunitate (sau de compania care dezvoltă distribuția)
- familiaritatea cu anumite componente ale sistemului, în principal sistemul de pachete (acesta nu reprezintă un criteriu foarte important pentru noii utilizatori)
- suportul pentru hardware-ul disponibil în calculator – distribuțiile pot fi disponibile pentru platforme diferite (procesoare diferite) și pot oferi acces facil la driver-e pentru hardware variat
- resursele disponibile pe calculator – în cazul în care resursele calculatorului pe care va instala Linux sunt scăzute, se recomandă căutarea unei distribuții cu cerințe scăzute

[1] Redundant Array of Inexpensive Disks
[2] Logical Volume Manager

- destinația sistemului de operare – în cazul în care se urmărește utilizarea Linux pentru o anumită activitate/sarcină se recomandă instalarea unei distribuții special concepute (în cazul în care există) – (vezi secțiunea 2.1.1).

Distribuția aleasă ca suport pentru această carte este *Kubuntu 9.10 Desktop*[1], distribuție derivată din Ubuntu, care:

- oferă un nivel acceptabil de stabilitate raportat la nivelul de noutate al pachetelor
- are o comunitate foarte largă, formată din toți utilizatorii de Ubuntu și derivate ale acestuia
- oferă suport facil pentru hardware proprietar, hardware care nu are driver-e open-source disponibile implicit în distribuțiile Linux; majoritatea dispozitivelor comune sunt suportate
- poate funcționa bine pe un sistem desktop actual, dat fiind cerințele acesteia[2]
- sistemul pe care va fi instalat va fi utilizat ca desktop

2.2.2 Pregătirea sistemului

Verificarea cerințelor sistemului. Spațiul de swap

Fiecare sistem de operare (și implicit distribuție Linux) este însoțit de o serie de cerințe minime hardware. Acestea sunt recomandări care, în cazul în care sunt îndeplinite, asigură buna funcționare a sistemului de operare. În anumite situații, programele care se ocupa de instalarea sistemului de operare vor detecta neîndeplinirea cerințelor minime hardware și vor refuza contiuarea procesului de instalare.

Distribuția aleasă în această carte (Kubuntu 9.10) are următoarele cerințe hardware (recomandări, în paranteză se regăsesc cerințele minime):

- procesor x86 700 MHz (min: 300 MHz)
- 384 MB RAM (min: 64 MB RAM – la instalarea fără mediu grafic)
- 8 GB spațiu pe HDD (min: 4 GB)
- placă video capabilă să afișeze rezoluția 1024x768 (min: 640x480)
- pentru instalare și actualizare a sistemului – conexiune la Internet (min: CD-ROM sau placă de rețea)

Printre principalele probleme ce țin de utilizarea facilă a unui sistem de operare o reprezintă lipsa memoriei RAM. Sistemul de operare folosește pentru stocarea datelor temporare memoria RAM. Aceasta are avantajul unui timp de răspuns și al unei viteze de citire foarte mari, însă este limitată din punctul de vedere al capacității (sistemele actuale nu au, în mod normal, mai mult de 2GB de memorie RAM).

Atunci când întreaga memorie RAM este umplută cu date, există două soluții alternative. Prima din ele constă în nealocarea de memorie RAM pentru programele

[1] http://www.kubuntu.org/

[2] https://help.ubuntu.com/community/Installation/SystemRequirements

care cer acest lucru. Ca urmare programul respectiv nu va putea funcționa și va genera erori. Atunci când programul în cauză este unul vital pentru funcționarea sistemului de operare, funcționarea întregului sistem de operare poate deveni instabilă.

A doua variantă este aceea în care datele care ar trebui stocate în memoria RAM sunt stocate într-o alta memorie (pe hard-disk sau pe o memorie pe USB). Deoarece memoria RAM este cea mai rapidă memorie din sistem, utilizarea unui alt tip de memorie va atrage după sine penalizări de performanță. Această soluție se numește **swapping** datorită modului în care funcționează: dacă memoria RAM este plină și apar cereri pentru alocarea de noi zone de memorie, o parte din datele aflate deja în memoria RAM sunt mutate pe un alt tip de memorie, în acest fel eliberându-se memorie RAM. În momentul în care datele mutate anterior pe un alt tip de memorie sunt cerute de programe, ele sunt mutate înapoi în RAM (daca nu este suficient loc în acel moment se va elibera o parte din RAM copiind alte date din RAM pe suportul de memorie).

Cea mai uzuală și ieftină soluție de memorie folosită pentru swap este hard-disk-ul (vezi secțiunea 5.6). Sistemele de operare de tip Linux folosesc în general o zonă dedicată pe hard-disk numită **partiție de swap**. Această partiție are în general dimensiuni apropiate de cele ale memoriei RAM (de ordinul 1-2 GB). Windows-ul și sistemul de operare Mac OS X folosesc pentru swap fișiere a căror dimensiune se poate extinde sau micșora în funcție de necesități.

Partiționare

O **partiție** reprezintă o diviziune logică a unui hard-disk. Prin partiționare un hard-disk este împărțit în mai multe partiții. Pe un hard-disk poate exista și o singură partiție.

Un hard-disk este **nepartiționat** atunci când pe el nu există nicio partiție.

Partiționarea unui hard-disk îl face pe acesta să aibă aceeași comportare ca mai multe hard-disk-uri independente. Diferența între partiții și hard-disk-uri reale constă în faptul că partițiile de pe un hard-disk folosesc aceleași resurse hardware (și, implicit, în cazul unei defecțiuni hardware a hard-disk-ului, datele de pe toate partițiile pot deveni inaccesibile). În plus partițiile pot fi redimensionate după ce au fost create, în schimb discurile au o dimensiune fixă.

Într-o partiție se poate crea un sistem de fișiere pentru stocarea fișierelor sau partiția poate fi utilizată în scopuri speciale, cum sunt de exemplu partițiile de swap. Partițiile de swap nu folosesc un sistem de fișiere ca NTFS, FAT32 sau ext3. Sistemul de operare are metode proprii de a organiza datele în ele și de a le face ușor de accesat (vezi capitolul 4).

Avantajele utilizării de partiții sunt următoarele:

- posibilitatea instalării mai multor sisteme de operare pe același hard-disk – se poate alege ulterior, la pornirea sistemului de calcul, care din ele va fi folosit (vezi secțiunea 6.2);
- posibilitatea de a separa datele personale de datele folosite de sistemul de operare și de programe – dacă datele personale ale utilizatorilor se găsesc pe o

altă partiție, sistemul de operare nu mai poate fi pornit și este necesară reinstalarea lui, datele nu vor fi afectate în niciun fel;

- se poate utiliza același spațiu de swap între mai multe sisteme de operare de tip Linux instalate pe același sistem de calcul (folosind aceeași partiție de swap pentru toate).

Tipuri de partiții

Primul sector fizic de pe un hard-disk poartă numele **Master Boot Record (MBR)** (vezi secțiunea 6.2.1). Acest sector are 512 octeți și conține informații importante despre modul în care este partiționat hard-disk-ul.

Dacă acest sector este șters sau suprascris toate informațiile de pe hard-disk se pierd.[1]

În cadrul MBR se găsesc informații despre fiecare partiție de pe hard-disk. Aceste informații sunt stocate într-o tabelă de partiții primare care are doar patru intrări. Prin urmare MBR-ul poate reține informații pentru doar patru partiții primare. Una dintre aceste patru partiții va fi marcată ca activă și de pe ea se va încerca pornirea sistemului de operare.

O **partiție primară** poate conține doar un singur sistem de fișiere. Pentru a elimina limitarea pe care o prezintă MBR-ul (de a putea reține doar patru partiții primare) a fost creată **partiția extinsă**. Acest tip de partiție ocupă tot o intrare în tabela de partiții din MBR însă poate conține la rândul său mai multe **partiții logice**, fiecare partiție având asociat un sistem de fișiere. Numărul de partiții logice incluse într-o partiție extinsă este limitat doar de dimensiunile partiției extinse și de structura de date folosită pentru a reține informațiile despre partițiile logice.

Unele sisteme de operare trebuie să fie instalate pe partiții primare pentru a putea porni. În această categorie intră MS-DOS și toate versiunile de Windows care depind de el (Windows 3.x, Windows 95, Windows 98, Windows Millenium), precum și BSD.

2.2.3 Live CD

Pentru Ubuntu 9.10 (și derivatele precum Kubuntu) varianta grafică de instalare este disponibilă sub forma unui Live CD.

Un **Live CD** (sau Live DVD) este un CD (DVD) care conține un sistem de operare care poate fi pornit direct (boot-abil), fără a avea nevoie de instalarea pe un hard-disk sau de prezența acestuia.

Există și Live USB-uri, dispozitive USB boot-abile cu sisteme de operare care au, de multe ori, adăugată și posibilitatea scrierii informațiilor pe USB (comparativ cu Live CD/DVD-urile unde informațiile se reține implicit doar în memorie).

[1] chiar dacă informațiile din MBR se pierd, folosind utilitare specializate se pot recupera datele de pe hard-disk prin recunoașterea tiparelor caracteristice sistemelor de fișiere

Termenul "Live" vine de la faptul că pe un singur disc se găsesc toate fișierele necesare funcționării unui sistem de operare, fără a fi necesară copierea unui fișier pe hard-disk înaintea pornirii sistemului de operare.

Astfel un Live CD nu modifică în niciun fel sistemul de operare instalat pe hard-disk, excepție făcând situațiile când utilizatorul dorește în mod expres acest lucru. Un sistem de operare prezent pe un Live CD este totuși unul normal, astfel încât are nevoie de existența unui suport pentru a putea salva informații de stare în fișiere etc.

> Pentru a rezolva problema lucrului cu fișiere, în condițiile în care în mod implicit nu se lucrează cu un hard-disk, un Live CD folosește un **ramdisk** (o porțiune din memoria RAM) precum un hard-disk.

Toate datele care se scriu în *ramdisk* sunt accesibile pe toată perioada în care sistemul de operare rulează. Atunci când acesta este oprit, datele din RAM se pierd și, implicit, tot conținutul *ramdisk*-ului este șters. De aceea, la oprirea unui sistem pornit de pe Live CD, sistemul de calcul revine la starea anterioară pornirii.

Utilizarea unei porțiuni din memoria RAM pe post de hard-disk oferă avantajul unui timp de acces și a unei rate de transfer mai bune decât în cazul unui disk. Dezavantajul este acela că aplicațiile vor avea disponibilă o cantitate mai mica de RAM. Pentru a putea rula în condiții normale un Live CD are nevoie de 512MB de memorie RAM, minimul necesar fiind 256MB.

Așadar un Live CD oferă posibilitatea testării și utilizării unui sistem de operare în condiții de deplină siguranță pentru software-ul instalat pe sistemul de calcul. De multe ori, Live CD-urile sunt folosite pentru recuperarea datelor aflate pe o partiție ce nu mai poate fi accesată din sistemul de operare instalat pe hard-disk (spre exemplu atunci când sistemul de operare nu mai porneste).

2.2.4 Instalare Kubuntu

Kubuntu este o distribuție derivată din Ubuntu, distribuție care folosește ca desktop environment KDE în loc de GNOME (vezi secțiunea 13.7). Kubuntu este parte a proiectului Ubuntu și folosește aceeași arhitectură, aceleași repository-uri și aceleași pachete ca și Ubuntu (vezi secțiunea 3.2).

Pentru a instala Kubuntu (și în general orice distribuție Linux) pe hard-disk este necesară crearea a cel puțin două partiții: una pe care va fi instalat sistemul de operare și una pentru swap. Acest lucru poate fi realizat înaintea instalării propriu-zise (folosind programe specializate în administrarea partițiilor cum ar fi *fdisk*, *GParted*, *PQMagic* etc.) sau poate fi realizat în timpul instalării folosind utilitarul de partiționare pus la dispoziție în acest sens de programul asistent de instalare.

Instalarea Kubuntu 9.10 (varianta Desktop) începe prin introducerea Live CD-ului în unitatea optică și repornirea calculatorului (alternativ, se poate realiza instalarea de pe USB stick). În cazul în care BIOS-ul nu este configurat să inițializeze sistemul de operare de pe unitatea optică (USB stick), trebuie realizată această configurare.

După ce calculatorul a inițializat sistemul de operare de pe CD, va fi afișat un ecran ce permite alegerea modului în care va fi folosit Live CD-ul (figura 2.2)

Figura 2.2: Alegerea modului în care va fi folosit Live CD-ul

Prima opțiune din listă va porni Kubuntu folosind configurările implicite. Din mediul grafic pornit se poate alege ulterior pornirea instalării. A doua opțiune (*Install Kubuntu*) va porni direct instalarea. *Check disc for defects* este util pentru verificarea CD-ului. *Memory test* va porni un test al memoriei RAM, util pentru depanarea problemelor hardware atunci când memoria RAM este suspectată că ar fi defectă.

După ce este aleasă prima opțiune sistemul de operare va fi inițializat. Desktp-ul implicit din Kubuntu este prezentat în figura 2.3.

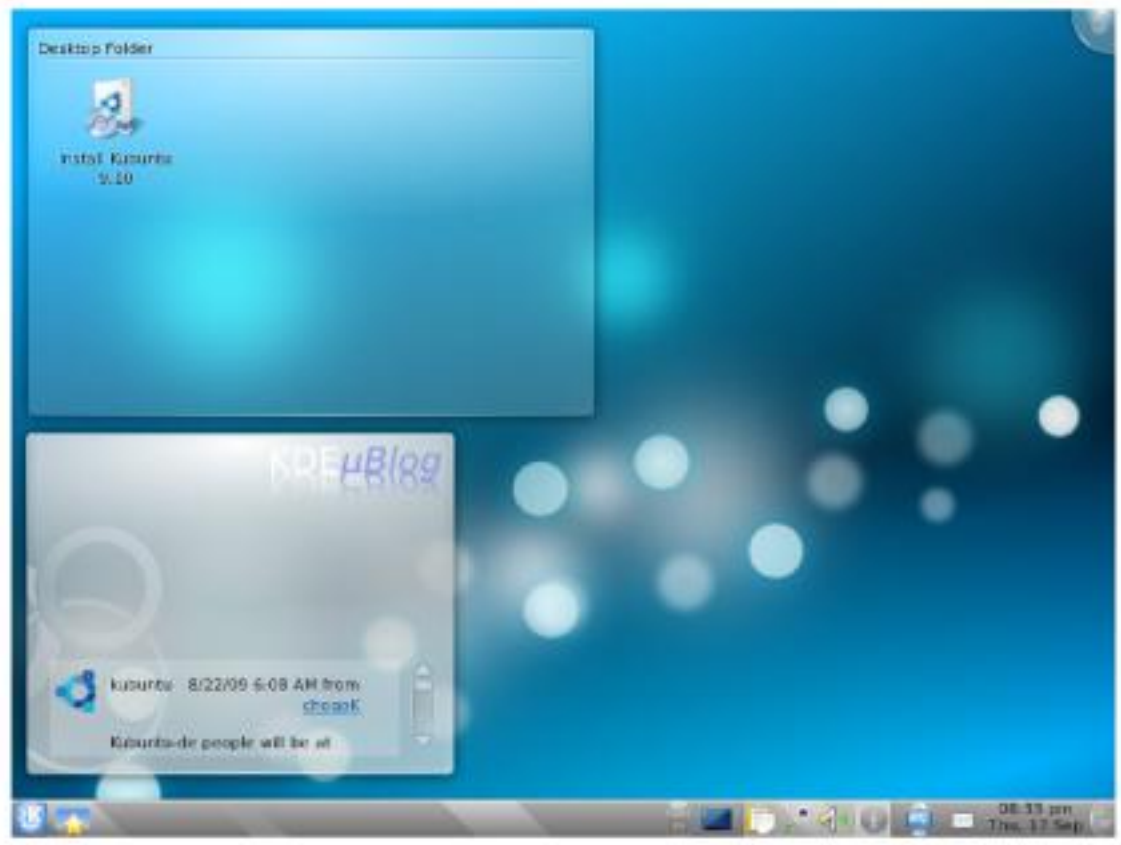


Figura 2.3: Desktop-ul implicit al Live CD-ului Kubuntu

În acest moment Kubuntu poate fi folosit ca Live CD. Pentru a îl instala pe hard-disk, se va accesa pictograma *"Install Kubuntu 9.10"* de pe desktop. Rezultatul va fi pornirea unui asistent de instalare (figura 2.4).

Figura 2.4: Alegerea limbii folosită de asistentul de instalare

Prima etapă constă în alegere limbii folosită de asistentul de instalare. Etapa următoare permite alegerea locului în care se găsește utilizatorul (vezi figura 2.5). Această configurare are efect asupra fusului orar ce va fi stabilit pentru calculator.

Figura 2.5: Alegerea locului în care se află utilizatorul

Pasul următor este reprezentat de alegerea felului în care sunt amplasate tastele pe tastatură (*keyboard layout*) (figura 2.6). După ce se realizează această configurare urmează pasul cel mai important din cadrul instalării: partiționarea hard-disk-ului (vezi figura 2.7)

Discul poate fi șters integral, caz în care datele deja existente vor fi șterse, sau poate fi partiționat manual, alegând dimensiunea partițiilor și tipul lor. În continuare va fi prezentată partiționarea manuală.

Figura 2.6: Alegerea felului în care sunt amplasate tastele pe tastatură

Figura 2.7: Partiționarea manuală a hard disk-ului

În cazul în care discul este nepartiționat este necesară crearea unei noi tabele de partiții apăsând butonul *New Partition table*. Rezultatul va fi afișarea tabelei de partiții, prezentată și în figura 2.8.

În cazul în care exista deja partiții pe disc, acestea vor fi afișate în tabela de partiții. În continuare, se presupune că hard-disk-ul nu este partiționat și se vor crea două partiții: una *ext4* și una pentru *swap*. Pentru a realiza acest lucru se apasă butonul *Add...* din fereastra care prezintă tabela de partiții (figura 2.9).

După cum se observă în figura 2.9, partiția *ext4* este primară, are 4096 MB, este creată la începutul spațiului liber și va fi accesată de sistemul de operare prin directorul / (*mount point*).

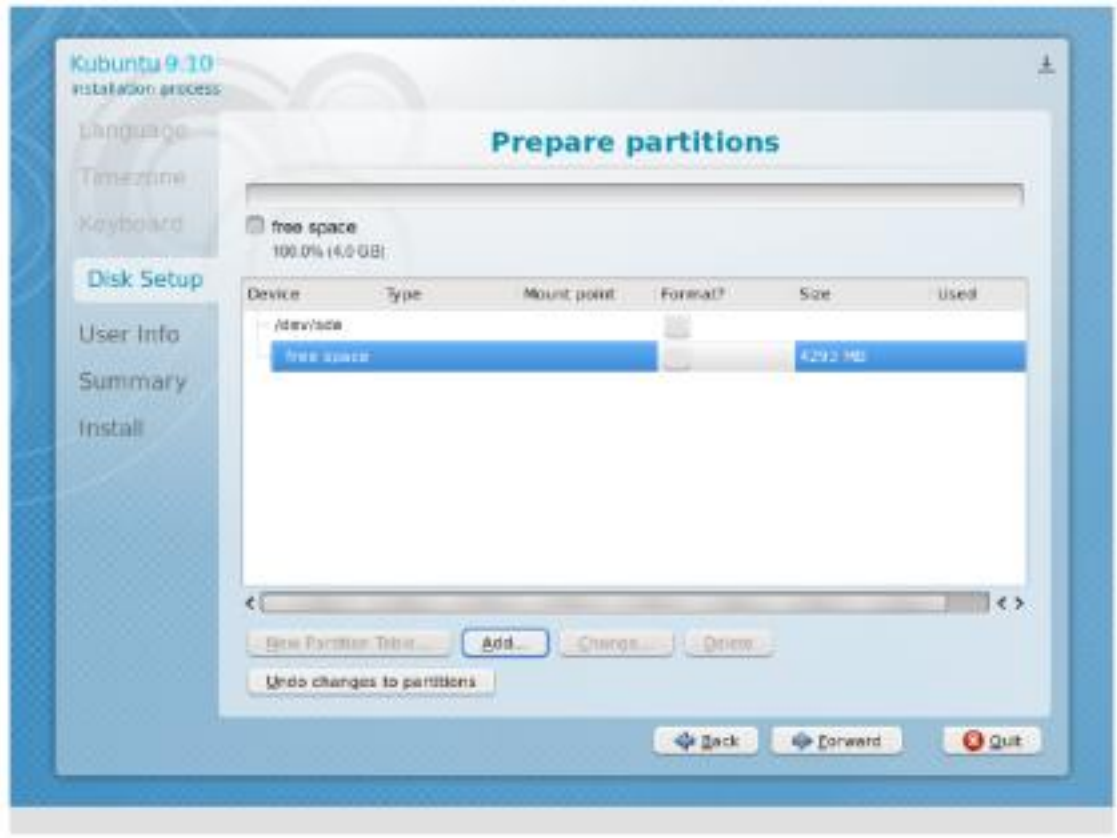


Figura 2.8: Tabela de partiții – inițial discul nu are nicio partiție

Figura 2.9: Crearea unei partiții

Figura 2.10: Crearea unei partiții swap

Partiția de swap (figura 2.10) este tot primară, are 197 MB și este creată la începutul spațiului liber rămas. Ea nu va putea fi accesată printr-un director, ci va fi folosită de sistemul de operare în mod transparent.

Această schemă de partiționare este pur demonstrativă. După crearea celor două partiții, tabela de partiții arată ca în figura 2.11.

După crearea partițiilor, trebuie datele personale ale utilizatorului: numele lui, numele de utilizator dorit, parola și numele stației (*hostname*). Asistentul de instalare mai permite și alegerea modului de autentificare în sistem și dacă directorul cu datele utilizatorului

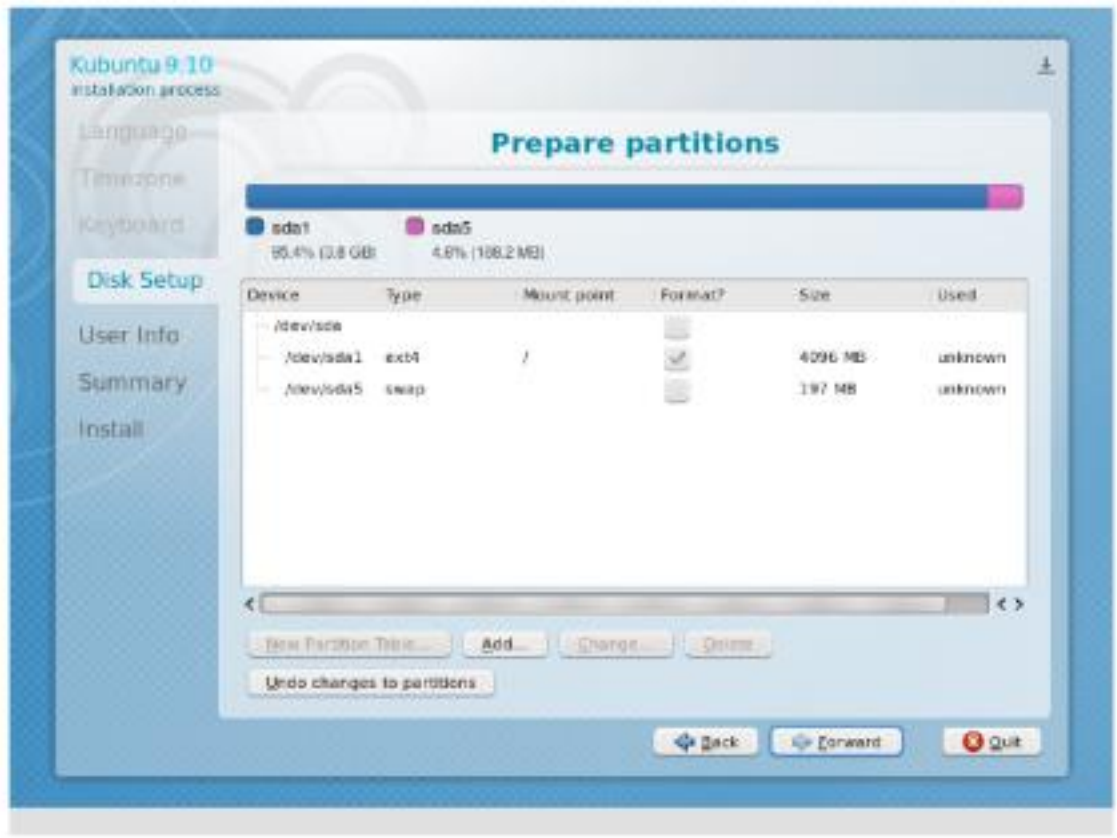


Figura 2.11: Tabela de partiții după ce au fost create cele două partiții

va fi criptat (vezi figura 2.12) (vezi secțiunea 10.5.2).

Figura 2.12: Configurarea datelor personale ale utilizatorului

În pasul următor (ultimul pas al instalării) pot fi revizuite configurările realizate. Apăsând pe butonul *Advanced...* se poate configura unde va fi instalat *bootloader*-ul (vezi secțiunea 6.2) bootloader/GRUB (vezi figura 2.13).

Pentru a începe copierea fișierelor pe hard-disk se apară butonul *Install*. La terminarea instalării va fi afișat mesajul din figura 2.14.

După închiderea asistentului de instalare sistemul de operare va rula în continuare ca Live CD. Pentru a porni sistemul de operare proaspăt instalat, calculatorul trebuie repornit.

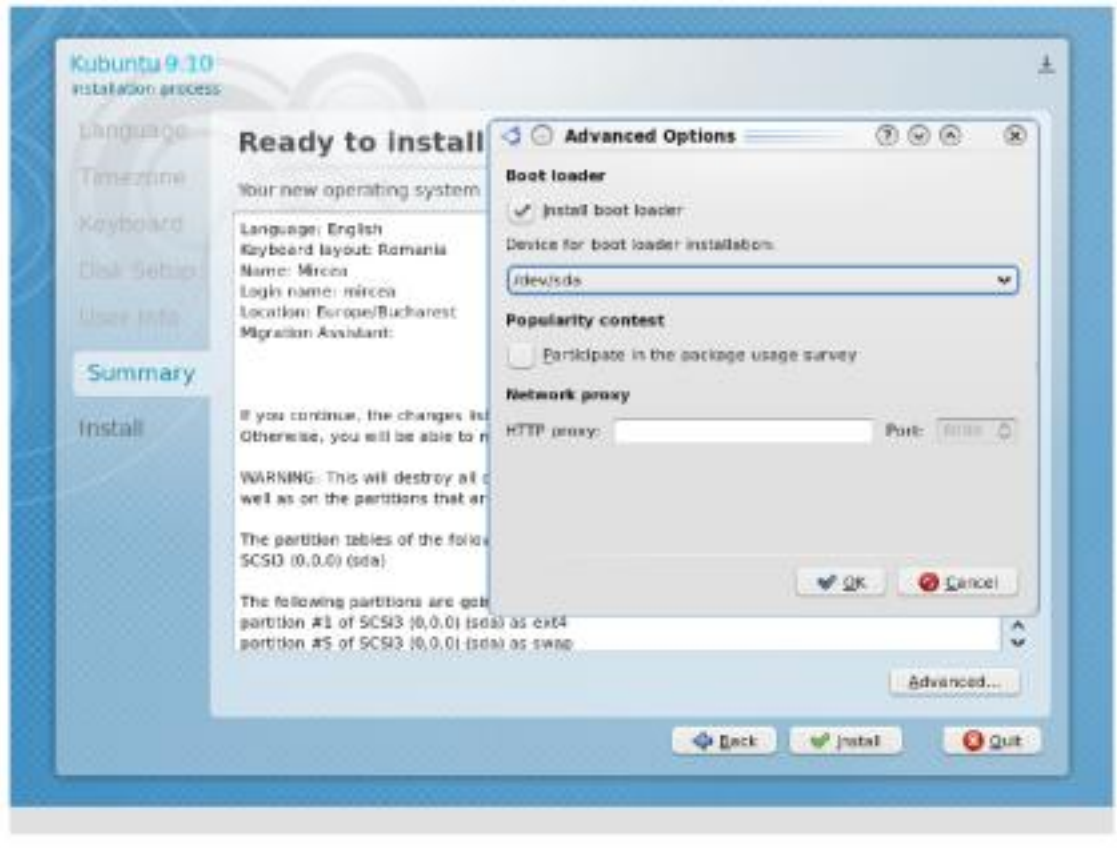


Figura 2.13: Verificarea datelor și alegerea hard-disk-ului pe care va fi instalat GRUB

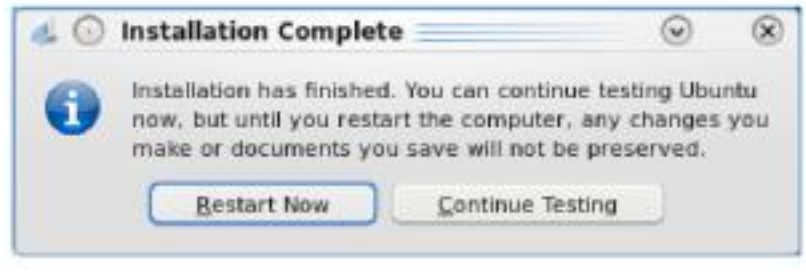


Figura 2.14: Mesajul care marchează instalarea cu succes a sistemului de operare

2.3 Interacțiunea cu sistemul de operare

2.3.1 Interfețele cu utilizatorul

Primele interfețe cu utilizatorul au fost, din motive tehnice, create pe monitoare care afișau doar caractere text. De aceea, posibilitățile acestor interfețe de a crea componente grafice au fost limitate. În această categorie (interfețe care utilizează doar caractere în mod text) intră CLI (*Command Line Interface*) și TUI (*Text User Interface*). În timp, atât monitoarele cât și plăcile video au avansat tehnologic și au apărut astfel afișajele grafice (cu rezoluții mai mari decât cele text). În acest fel a apărut și o nouă clasa de interfețe cu utilizatorul numită GUI (*Graphical User Interface*).

În **CLI**, interacțiunea cu sistemul de operare se bazează pe comenzi scrise de la tastatură. Exemple de CLI pot fi văzute în figura 2.17 și în figura 2.18.

Avantajul acestei metode este acela că pot fi invocate comenzi complexe utilizând secvențe scurte de caractere. Pentru a realiza sarcini echivalente într-un GUI poate fi uneori nevoie și de câteva zeci de click-uri de mouse. Un alt avantaj al CLI este rapiditatea dobândită în utilizare, după ce comenzile au fost reținute (în special dacă este folosit istoricul de comenzi: prin apăsarea tastei `SăgeatăSus` pot fi reexecutate comenzile date anterior). Luând ca exemplu configurarea unei adrese IP (vezi secțiunea 8.2.1) pe o interfață de rețea, diferența de viteză între prima și a zecea oară când se realizează această configurare într-un GUI este mica, datorită numărului mare

de click-uri necesar. În cazul unui CLI, folosind și istoricul de comenzi, diferența de viteză este extrem de mare.

Dezavantajul principal al CLI este curba abruptă de învățare, utilizatorul obișnuindu-se mai greu cu o comandă decât cu reținerea locului în care se află un buton.

Deși de la apariția sa a fost concurat de TUI și GUI, CLI rămâne metoda de control al sistemului de operare preferată de utilizatorii avansați datorită eficienței în utilizare.

În **TUI**, interacțiunea cu sistemul de operare se realizează prin intermediul ferestrelor, meniurilor, butoanelor și mouse-ului. Pentru desenarea interfeței sunt utilizate caractere disponibile pe orice terminal text (figura 2.15).

Figura 2.15: Text User Interface

Acest tip de interfață se apropie de modul în care arată o interfață grafică însă, datorită faptului că folosește caractere text, rezoluția ecranului este mai mică.

GUI este un sistem de interacțiune cu utilizatorul de tipul WIMP (*window, icon, menu, pointing device*) (figura 2.16).

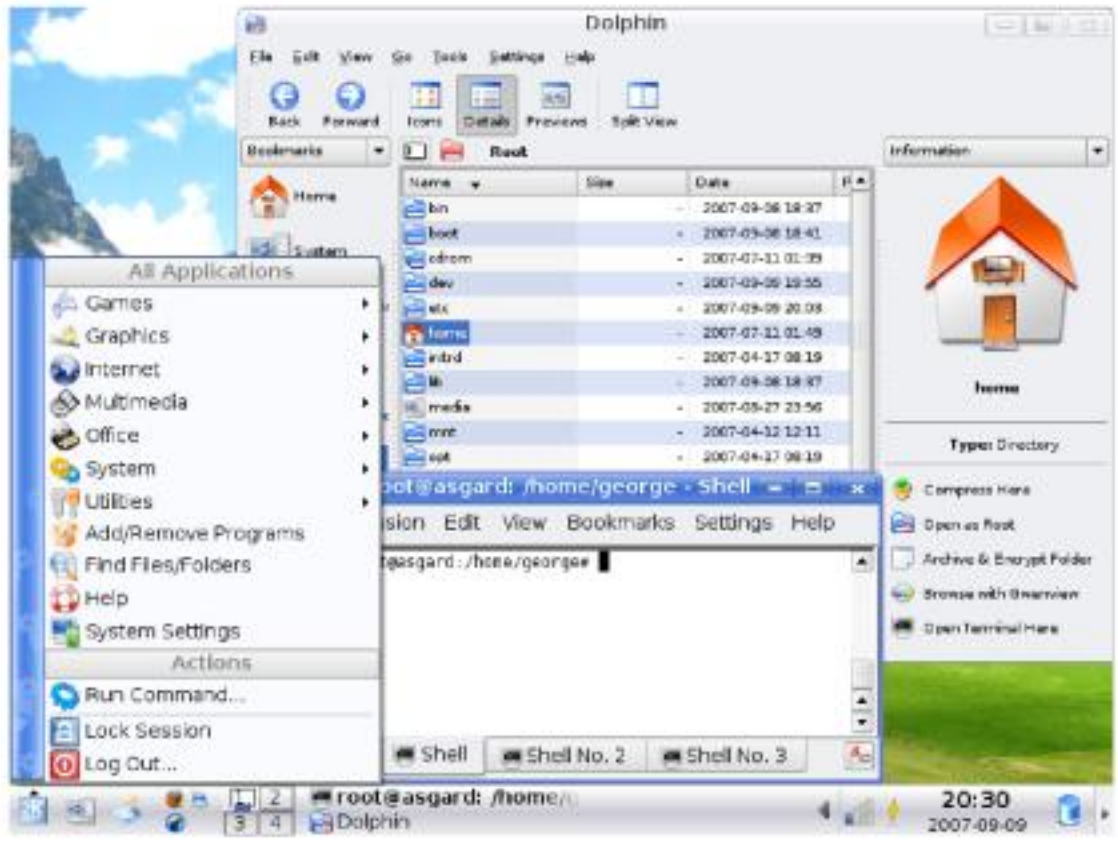


Figura 2.16: Graphical User Interface

Pentru desenarea interfeței GUI sunt folosite moduri grafice de rezoluție înaltă, cum ar fi VGA=640x480, SVGA=800x600, 1024x768, 1280x1024, 1600x1200 etc.

2.3.2 Interfața în linie de comandă (CLI)

Consola

> O **consolă** este un tip de echipament care permite introducerea și afișarea de text folosit în administrarea și utilizarea sistemului de operare.

Inițial consolele erau formate dintr-un terminal (un monitor ce avea atașată o tastatură) care se conecta la sistemul de calcul printr-o conexiune serială. Acest tip de console este foarte rar întâlnit în ziua de astăzi.

Pe un calculator obișnuit, monitorul împreună cu tastatură au același rol ca un terminal. Ele permit, în Linux, accesarea a două tipuri de console: *terminalele virtuale* (console în mod text) și *emulatoarele de terminale* (console în mod grafic).

Terminalele virtuale au o rezoluție mică (în general 25 de linii cu 80 de caractere pe linie). Ele pot fi accesate din interfață grafică cu ajutorul tastelor `CTRL+ALT+Fx`, unde Fx reprezintă tastele F1 până la F6 (în mod implicit). Din aceste terminale virtuale se poate reveni în interfață grafică cu ajutorul combinației de taste `(CTRL+)ALT+F7`.

Trecerea dintr-un terminal virtual în altul poate realiza prin trei combinații de taste. Primele două sunt: `CTRL+ALT+Fx` și `ALT+Fx` (unde x este numărul terminalului în care se va trece). A treia combinație de taste este `ALT+Sageată Stânga` sau `ALT+Sageată Dreapta` și permite trecerea dintr-un terminal în cel precedent, respectiv în cel următor. De exemplu trecerea din terminalul 4 în terminalul 3 se poate realiza cu `CTRL+ALT+F3` sau `ALT+F3` sau `ALT+Sageată Stânga`.

Valorile tastelor `F` prezentate sunt cele implicite pe majoritatea distribuțiilor. Aceste valori pot fi schimbate. De exemplu, interfața grafică poate fi plasată pe terminalul trei sau pe terminalul cinci; numărul de terminale accesibile poate fi șase (valoarea implicită) sau zece (figura 2.17).

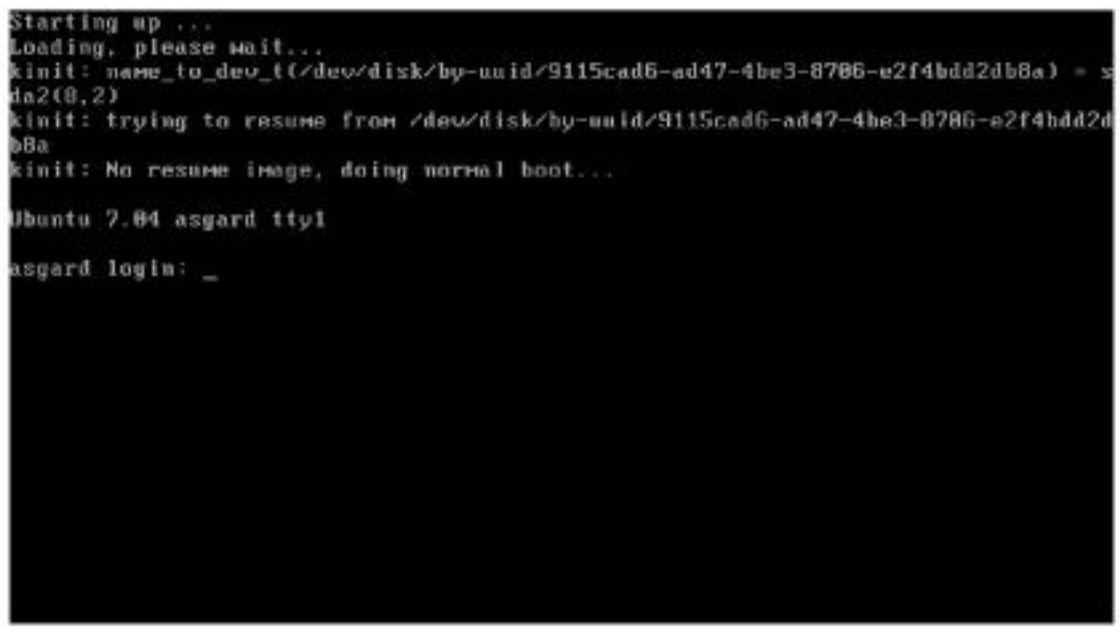


Figura 2.17: Terminal virtual (CLI)

Consolele în mod grafic reprezintă **emulatoare de terminal** (figura 2.18). Dimensiunea lor pot fi extinsă peste cea de 25x80 a terminalelor virtuale. De asemenea, un utilizator poate porni un număr nelimitat de astfel de emulatoare. Cele mai cunoscute emulatoare sunt **`konsole`** (pentru KDE), **`gnome-terminal`** (pentru GNOME) și **`xterm`**.

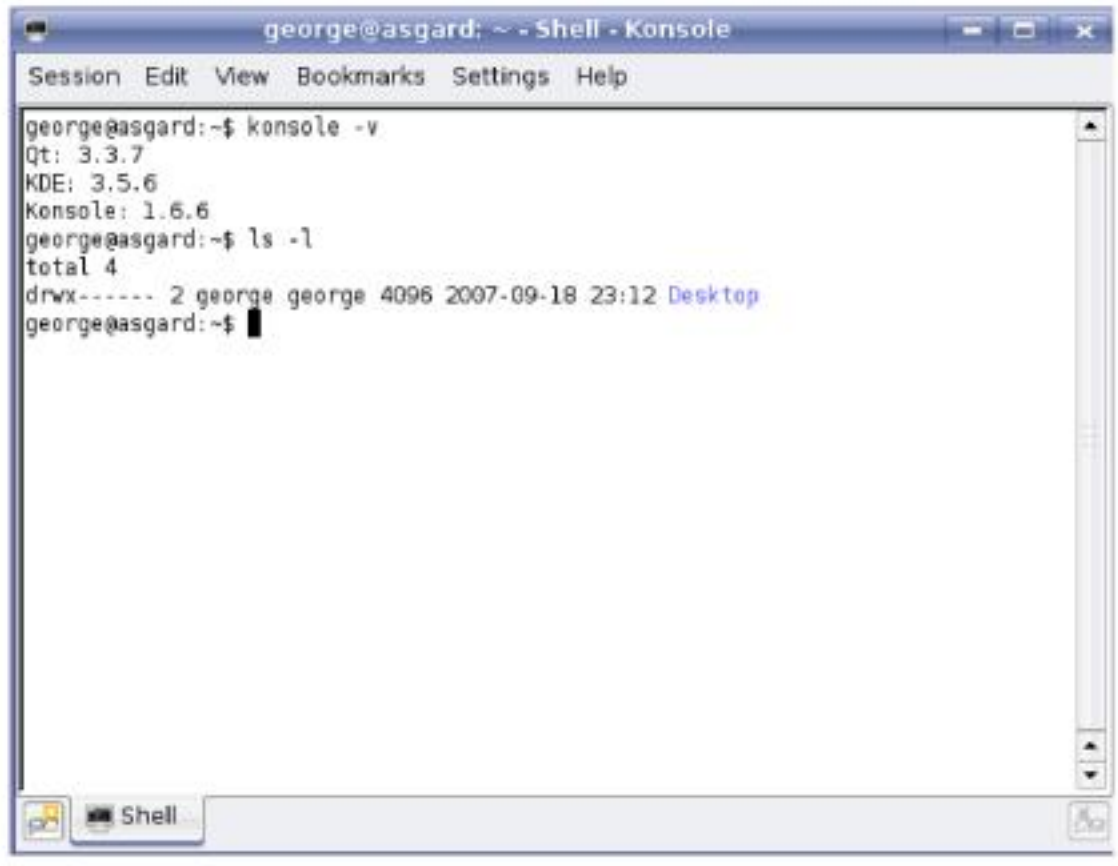


Figura 2.18: Emulator consolă (CLI)

Atunci când un utilizator accesează un terminal virtual el trebuie să se autentifice folosind un nume de utilizator și o parolă. La pornirea unui emulator de terminal utilizatorul nu mai trebuie să se autentifice, deoarece autentificarea s-a produs înainte de pornirea interfeței grafice. De asemenea, la pornirea unui emulator de terminal utilizatorul curent va fi cel care este autentificat în interfața grafică.

În ultimii ani, îndepărtarea de linia de comandă a sistemului de operare Linux este din ce în ce mai accentuată în detrimentul utilizării interfeței grafice, mai intuitive și cu facilități mult mai avansate. Cu toate acestea, interfața în linia de comandă rămâne o componentă esențială în configurarea rapidă a unui sistem și a administrării de la distanță. Interfața în linia de comandă permite toate operațiile care pot fi realizate și din interfața grafică.

Shell

În IT, denumirea de *shell* are mai multe sensuri. Unul dintre acestea este sensul de componentă software care realizează interfața cu utilizatorul. Această interfață poate fi una CLI, caz în care *shell*-ul este un interpretor de comenzi, sau una GUI, caz în care *shell*-ul este o interfață grafică dintre utilizator și resursele sistemului de operare (cum ar fi Explorer în Windows).

Comenzile Linux sunt transmise la o consolă. Consola (indiferent de tipul ei – emulator de terminal sau terminal virtual) rulează un shell, shell care permite introducerea

comenzilor. Există mai multe aplicații shell de interpretare a comenzilor în Linux, cea mai uzuală fiind denumită **bash** (Bourne Again SHell).

Prompt-ul

Shell-ul (interpretorul) de comenzi oferă utilizatorului un prompt. Acesta are un format de genul:

```
1 username@localhost:~$
```

La acest prompt, utilizatorul poate introduce comenzi. `username` reprezintă numele utilizatorului, iar `localhost` este numele stației (*hostname*). Simbolul `~` indică directorul curent, home-ul utilizatorului `george`[1]. Simbolul `"$"` marchează terminarea prompt-ului și începerea zonei de introducere a comenzilor. De exemplu, în cazul următor:

```
1 george@asgard:~$
```

`george` este numele utilizatorului și `asgard` este numele stației.

Pentru mai multe informații despre interacțiunea cu shell-ul consultați secțiunea 12.2.

Comenzi

Comenzile sunt cuvinte cheie care se introduc la consolă (și interpretate de interpretorul de comenzi), folosite pentru a configura sistemul de operare sau pentru a obține informații de la acesta.

Comenzile pot fi simple sau pot avea parametri. O **comandă simplă** este o comandă care conține un singur cuvânt cheie suficient pentru a executa o acțiune. Majoritatea comenzilor sunt însă **comenzi compuse** și pot primi unul sau mai mulți parametri care pot afecta rezultatul comenzii. Separarea între numele comenzii și parametru sau între doi parametri se realizează prin intermediul caracterului "spațiu". De multe ori, primul parametru (sau și alți parametri, în funcție de caz) este marcat prin caracterul "-".

Mai jos este prezentat un exemplu de comandă simplă și apoi aceeași comandă cu parametri. Comenzile au de obicei rezultate, rezultate care sunt afișate de către interpretorul de comenzi (shell) pe consola utilizatorului. În exemplul nostru, fiecare comandă are un rezultat care este afișat.

```
1  george@asgard:~$ ps
2    PID TTY          TIME CMD
3  13391 pts/0    00:00:00 bash
4  13666 pts/0    00:00:00 ps
5
6  george@asgard:~$ ps -o uid,pid,cmd
7    UID   PID CMD
8   1000 13391 -bash
9   1000 13669 ps -o uid,pid,cmd
10
11 george@asgard:~$
```

[1] detalii despre ierarhia sistemului de fișiere se găsesc în secțiunea 4.1.2

Setul cel mai important de comenzi este cel care interacționează cu sistemul de fișiere. Aceste comenzi sunt fundamentale pentru interacțiunea cu sistemul și rularea altor comenzi importante.

Căutarea de ajutor

Daca se dorește căutarea de informații în legătură cu o comandă (parametri, date de ieșire, funcționalitate etc.) distribuțiile de Linux pun la dispoziție mai multe opțiuni:

- Transmiterea parametrului `--help` unei comenzi pentru a afișa un sumar al parametrilor posibili ai acelei comenzi. De exemplu:

```
1  george@asgard:~$ cp --help
2  Usage: cp [OPTION]... [-T] SOURCE DEST
3    or:  cp [OPTION]... SOURCE... DIRECTORY
4    or:  cp [OPTION]... -t DIRECTORY SOURCE...
5
6  Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.
7
8
9  Mandatory arguments to long options are mandatory for short options
   too.
10
11   -a, --archive                same as -dpR
12       --backup[=CONTROL]       make a backup of each existing
   destination file
13   -b                           like --backup but does not accept an
   argument
14       --copy-contents          copy contents of special files when
   recursive
15   -d                           same as --no-dereference --preserve=link
16 [...]
```

- Comanda **`whatis`** este folosită pentru a afișa o scurtă descriere a unei comenzi:

```
1  george@asgard:~$ whatis ls
2  ls (1)               - list directory contents
3  george@asgard:~$ whatis pwd
4  pwd (1)              - print name of current/working directory
5  george@asgard:~$ whatis cp
6  cp (1)               - copy files and directories
7  george@asgard:~$ whatis poweroff
8  poweroff (8)         - stop the system.
```

- Comanda **`apropos`** afișează comenzile care au legătură cu cuvântul transmis ca parametru:

```
1  george@asgard:~$ apropos zip
2  bunzip2 (1)  - a block-sorting file compressor, v1.0.3
3  bzcmp (1)    - compare bzip2 compressed files
4  bzdiff (1)   - compare bzip2 compressed files
5  bzegrep (1)  - search possibly bzip2 compressed files for a regular
   expression
6  bzfgrep (1)  - search possibly bzip2 compressed files for a regular
   expression
7  [...]
```

- Comenzile **man** și **info** sunt utilizate pentru a afișa paginile de documentație complete asociate unei comenzi dorite. Aceste pagini conțin informații detaliate despre opțiunile posibile ale comenzilor și utilizările acestora. Comenzile **man** și **info** primesc ca argument numele comenzii a cărei documentație se dorește a fi afișată:

```
1 george@asgard:~$ man ls
2 Reformatting ls(1), please wait...
3
4 george@asgard:~$ man pwd
5 Reformatting pwd(1), please wait...
6
7 george@asgard:~$ info coreutils ls
8 george@asgard:~$ info coreutils cp
```

Autentificarea și părăsirea sistemului

Autentificarea locală în sistemul de operare se poate realiza folosind un terminal virtual sau se poate realiza în interfață grafică, printr-un *display manager* (vezi secțiunea 13.2.2).

În cazul în care utilizatorul s-a autentificat în interfață grafică, el poate părăsi sistemul de operare (*logout*) tot din mediul grafic, folosind opțiunea *Logout*.

În cazul în care utilizatorul s-a autentificat într-un terminal virtual, el poate părăsi sistemul de operare folosind comanda **logout** sau comanda **exit**.

Comanda **logout** este folosită pentru a părăsi un login shell. Acesta este un shell creat atunci când utilizatorul s-a autentificat în sistem.

Comanda **exit** este folosită pentru a părăsi atât un login shell cât și un shell creat din acest login shell (cu ajutorul comenzii **su** de exemplu).

2.3.3 Oprirea sistemului de calcul

Pentru a opri sau reporni un sistem de calcul se folosește comanda **shutdown**. Această comandă poate fi utilizată doar de către un utilizator cu drepturi administrative (vezi secțiunea 2.4.1). Sintaxa comenzii este:

```
1 shutdown [OPTION]... TIME [MESSAGE]
```

Printre opțiunile folosite în mod uzual se numără: `-r` pentru a reporni stația și `-h` pentru a o opri. `time` reprezintă momentul la care va fi oprită sau repornită stația, iar `message` este un mesaj ce va fi trimis pe toate terminalele legate la sistem.

De exemplu, pentru a opri stația imediat, se rulează comanda:

```
1 root@asgard:/home/george# shutdown -h now
2
3 Broadcast message from george@asgard
4         (/dev/pts/0) at 18:28 ...
5
6
7 The system is going down for halt NOW!
```

Pentru a reporni stația imediat, se rulează comanda:

```
root@asgard:/home/george# shutdown -r now
Broadcast message from george@asgard
        (/dev/pts/4) at 11:04 ...


The system is going down for reboot NOW!
```

Pentru a opri stația la ora 20:00:

```
root@asgard:/home/george# shutdown -r 20:00

Broadcast message from george@asgard
        (/dev/pts/4) at 11:03 ...


The system is going down for reboot in 537 minutes!
```

Pentru a opri stația în 10 minute de la executarea comenzii se folosește comanda:

```
root@asgard:/home/george# shutdown -r +10

Broadcast message from george@asgard
        (/dev/pts/0) at 18:33 ...

The system is going down for reboot in 10 minutes!
```

O altă posibilitate de a reporni stația este utilizarea comenzii **`restart`**. Pentru oprirea stației se pot folosi și comenzile **`halt`** sau **`poweroff`**. Toate aceste comenzi pot fi executate doar daca utilizatorul are cu drepturi administrative.

2.4 Configurări de bază ale SO

2.4.1 Administrarea sistemului Linux

În sistemele de operare Windows până la Windows XP (inclusiv), în timpul procesului de instalare se crea un utilizator cu drept de administrare. De cele mai multe ori, sistemul era folosit cu acest utilizator, situație care putea crea probleme de securitate datorită drepturilor extinse de care beneficia utilizatorul (vezi secțiunea 10.2). În sistemele de operare Linux se face o distincție implicită între drepturile de administrare și drepturile de utilizare a sistemului.

> În Linux, un singur utilizator are drepturi de administrare depline. Acesta este un utilizator special numit **`root`**.

Utilizatorul `root` poate realiza configurări, poate modifica modul în care pornește sistemul de operare, poate acorda drepturi parțiale altor utilizatori etc.

Pentru a putea realiza configurări sau acțiuni pe care doar utilizatorul `root` le poate realiza, un utilizator are trei posibilități.

Prima posibilitate o reprezinta autentificare în sistemul de operare folosind numele de utilizator `root` și parola asociată acestuia. În cele mai multe sisteme de operare folosite

în producție autentificarea ca `root` este dezactivată din motive de siguranță. De aceea această metoda poate să nu funcționeze.

Schimbarea între utilizatori (`su`)

În cazul în care utilizatorul este deja autentificat în sistem, există posibilitatea schimbării utilizatorulu (**a doua posibilitate** de obținere a drepturilor administrative). Din *shell*-ul în care se găsește deja, utilizatorul poate să pornească un nou *shell* în care să se autentifice ca un alt utilizator. Atunci când părăsește *shell*-ul nou creat (de exemplu prin comanda **`exit`**) el va reveni în *shell*-ul inițial.

Pornirea unui nou *shell* în care se realizeaze autentificarea ca alt utilizator se face cu ajutorul comenzii **`su`** (*switch user*). Această comandă poate primi ca parametru numele utilizatorului în contul căruia se va face autentificarea:

```
1 george@asgard:~$ su uso
2 Password:
3
4 uso@asgard:/home/george$ exit
5 exit
6
7 george@asgard:~$
```


Din motive de securitate, caracterele tastate în dreptul parolei nu vor fi afișate.

În exemplul anterior, utilizatorul autentificat în sistem este `george`. Utilizatorul `george` schimbă utilizatorul în `uso` folosind comanda **`su uso`** și se autentifică utilizând parola utilizatorului `uso`. Utilizatorul părăsește shell-ul nou folosind comanda `exit`, moment în care se observă revenirea acestuia în shell-ul inițial.

În cazul în care comanda **`su`** nu primește niciun parametru, implicit se va considera ca se dorește autentificarea ca utilizator `root`.

Ca root, execuția comenzii `su uso` va permite autentificarea ca utilizator `uso` fără introducerea parolei pentru acesta.

Pentru a părăsi un *shell* creat cu **`su`** nu se poate folosi comanda **`logout`**, deoarece sesiunea autentificată în sistem (login shell) aparține tot utilizatorului inițial (utilizatorul inițial fiind `george` în exemplul de mai jos) și nu noului utilizator (noul utilizator fiind `uso` în exemplul de mai jos):

```
1  george@asgard:~$ su uso
2  Password:
3
4  uso@asgard:/home/george$ logout
5  bash: logout: not login shell: use 'exit'
6
7  uso@asgard:/home/george$ exit
8  exit
9
10 george@asgard:~$
```

Ubuntu (și Kubuntu) au dezactivat în mod implicit contul de `root`, nefiind configurată nicio parolă pentru acesta. De aceea, autentificarea ca `root` atât la *prompt*-ul de login în sistem cât și cu **`su`** va eșua.

Execuția comenzilor cu drepturi administrative (sudo)

În cazul în care este dezactivată autentificarea ca `root`, **a treia posibilitate** de obținere a drepturilor administrative o reprezintă utilizarea comenzii **sudo**. Această comandă permite executarea unui program ca alt utilizator. Spre deosebire de **su**, care cerea parola utilizatorului al cărui cont va fi utilizat, **sudo** cere parola utilizatorului curent[1]. Din acest motiv comanda **sudo su** va avea succes (este cerută parola utilizatorului curent, nu cea a utilizatorului `root`, după care comanda **su** este executata ca `root`):

```
1 george@asgard:~$ sudo su
2 Password:
3
4 root@asgard:/home/george#
```

Atunci când utilizatorul `root` folosește comanda **sudo** pentru a porni un *shell* în care să se autentifice ca alt utilizator, nu i se va cere nicio parolă.

2.4.2 Asigurarea conectivității la Internet

Una dintre primele configurări ce trebuie realizată pe sistemul de operare instalat este asigurarea conectivității la Internet. Kubuntu este o distribuție de Linux care se bazează pe pachete și necesită o conexiune la Internet pentru a putea copia și instala versiuni mai recente ale acestora, realizând în acest fel actualizarea întregului sistem de operare.

Pentru asigurarea conectivității la Internet există mai multe alternative. Cea mai simplă variantă este aceea în care toți parametrii de rețea (vezi secțiunea 8.2) (adresă IP, mască de rețea, gateway, server DNS) sunt configurați în mod automat folosind DHCP[2]. Această metodă este cea încercată implicit de către sistemul de operare. Dacă aceasta reușește, utilizatorul are acces la Internet, nemaifiind necesară o altă configurare.

Dacă încercarea de configurare folosind DHCP eșuează, utilizatorul va trebui să configureze manual parametrii de rețea. Acest lucru se poate realiza *temporar* (efectele configurărilor vor dura până la următoarea pornire a sistemului de operare sau până la o nouă configurare) sau *permanent* (efectele configurărilor se vor păstra și după repornirea sistemului de operare).

Configurările temporare pentru adresa IP, masca de rețea și gateway se realizează folosind comenzile **ifconfig** și **route**. Pentru a putea folosi aceste comenzi utilizatorul trebuie să obțină drept de administrare. Pentru mai multe informații legate de configurarea rețelei consultați capitolul 8.

2.4.3 Actualizarea sistemului și a pachetelor

Printre primele acțiuni care se realizeaza imediat după instalarea unui sistem de operare se regăsește actualizarea acestuia. Într-un sistem bazat pe Debian (precum Ubuntu și Kubuntu), acest lucru se face prin execuția următoarelor comenzi ca `root`:

[1] acest comportament este dependent de distribuție – în Ubuntu și în distribuțiile derivate, comanda **sudo** cere parola utilizatorului curent

[2] Dynamic Host Configuration Protocol

```
root@asgard:~# apt-get update

root@asgard:~# apt-get dist-upgrade
```

Prima comanda actualizeaza lista de pachete cunoscută de către managerul de pachete, iar a doua comandă actualizează pachetele instalate pe baza informațiilor adunate de prima comandă. Pentru mai multe informații despre gestiunea pachetelor consultați secțiunea 3.2.

2.4.4 Administrarea utilizatorilor și a grupurilor de utilizatori

Majoritatea programelor de instalare a distribuțiilor Linux permit celui care realizează procesul de instalare să adauge direct utilizatori non-root în sistem, fără a fi nevoie de configurări ulterioare.

În cazul în care mai trebuie adăugați utilizatori sau grupuri, administratorul are la dispoziție comenzile **`adduser`** și **`addgroup`** pe sisteme Debian (și implicit Ubuntu/Kubuntu). **`adduser`** permite adăugarea în mod interactiv de utilizatori.

Dacă se dorește lucrul mai avansat cu utilizatori și grupuri (precum ștergerea, modificarea utilizatorilor/grupurilor) consultați secțiunea 3.1.

2.5 Studii de caz

2.5.1 Interoperabilitatea sistemului de fișiere între Linux și Windows

Atunci când pe un calculator este instalat și un sistem de operare Windows și unul Linux se pune de cele mai multe ori problema partajării de fișiere între cele două. Ca exemplu concret:

- cum se pot accesa din Linux (pentru scriere și citire) datele aflate în Windows
- cum se pot accesa din Windows datele aflate în Linux (pentru citire și scriere).

Soluția la această problemă se bazează pe ideea că pentru datele aflate pe cel puțin o partiție se va asigura suportul de citire și de scriere atât din Linux cât și din Windows. În acest sens există trei variante.

Prima dintre ele constă în utilizarea unei partiții FAT32 pentru a stoca datele ce se doresc a fi partajate. Avantajul acestei soluții este faptul că, atât în Windows cât și în Linux, există suport nativ pentru sistemul de fișiere FAT. Dezavantajele majore sunt limitarea dimensiunii unei partiții FAT32 la maxim 32GB și limitarea dimensiunii maxime a unui fișier la 4GB (ca atare pe o astfel de partiție nu se poate salva imaginea unui disc DVD de exemplu).

A doua soluție o reprezintă utilizarea unei partiții ext2 sau ext3 pentru stocarea datelor partajate. Această soluție nu prezintă limitările partiției FAT, însă Windows nu oferă suport nativ pentru acest tip de partiții. Pentru a putea accesa din Windows o partiție

ext2 sau *ext3* trebuie instalat un driver special. Acest driver se numește **Ext2 IFS**și este disponibil online[1].

Acest driver oferă suport de scriere și citire pentru sistemul de fișiere *ext2*. Sistemul de fișiere *ext3* este asemănător cu *ext2*, diferența dintre ele fiind reprezentată de suportul de jurnalizare pe care îl oferă *ext3*. *Ext2 IFS* poate citi și scrie date de pe o partiție *ext3*, însă nu va folosi suportul de jurnalizare pe care îl oferă aceasta.

Ext2 IFS nu funcționeaza cu sistemul de fișiere *ext4*, structurile de date aflate pe disc fiind diferite. Acest lucru trebuie luat în considerare în condițiile în care se dorește utilizarea acestei variante pentru partajare, mai ales când distribuțiile actuale au ca sistem de fișiere implicit *ext4* (vezi și secțiunea 4.7).

A treia variantă o reprezintă accesarea unei partiții NTFS din Linux. În Linux există suport pentru citirea datelor de pe un sistem de fișiere NTFS. Datorită faptului că standardul NTFS este unul închis, dezvoltarea unui driver care să ofere suport de scriere a fost greu de realizat. În momentul de față există însă un driver care oferă acest suport. El se numește *ntfs-3g*[2] și suport pentru acesta este oferit implicit în Ubunut/Kubuntu.

2.5.2 GParted

GParted este o aplicație de partiționare specifică Gnome[3] (figura 2.19). Aceasta poate crea, șterge, redimensiona, muta, verifica și copia partiții.

GParted oferă și o versiune disponibilă sub forma unui Live CD[4]. Aceasta este bazată pe distribuția Gentoo și permite administrarea partițiilor în cazul în care sistemul de operare nu mai pornește. De asemenea permite realizarea unei copii de siguranță a datelor în aceleași condiții.

Cuvinte cheie

- CLI
- consolă
- distribuție Linux
- gestionare utilizatori
- GUI
- Kubuntu
- Live CD
- MBR
- instalare
- mediu grafic
- partiție
- swap
- shell
- shutdown
- su
- sudo
- TUI

[1] http://www.fs-driver.org/

[2] http://www.ntfs-3g.org/

[3] Gnome este un Desktop Environment pentru mediul grafic din Linux – mai multe informații se pot găsi în secțiunea 13.7

[4] http://gparted.sourceforge.net/livecd.php

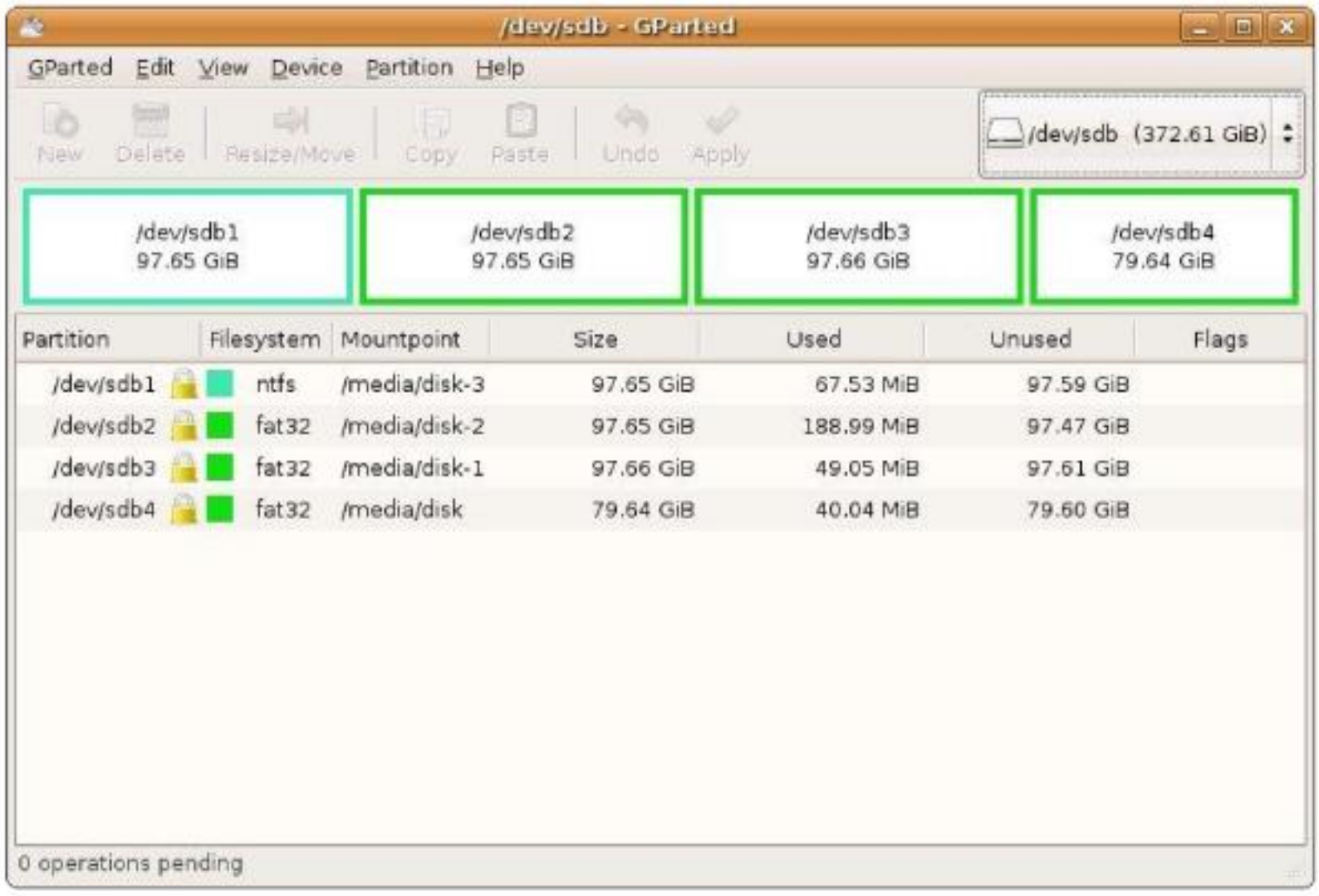


Figura 2.19: GParted

Întrebări

1. Câți utilizatori cu identificatorul 0 pot exista pe un sistem cu o partiție primară și două extinse:
   - ❑ niciunul
   - ❑ 1
   - ❑ 2
   - ❑ oricâți
2. Care dintre următoarele NU este necesară pentru a rula un sistem de operare de pe Live CD:
   - ❑ configurarea unei partiții primare
   - ❑ BIOS
   - ❑ RAM
   - ❑ procesor
3. Din ce sisteme de operare NU se pot realiza configurării din TUI?
   - ❑ Windows Vista
   - ❑ Debian
   - ❑ ubuntu Live CD
   - ❑ PalmOS

4. Operația de repornire a sistemului poate fi inițiată doar de utilizatorul privilegiat. În urma repornirii sistemului tot conținutul memoriei RAM este golit.
   - ❑ adevărat, adevărat
   - ❑ adevărat, fals
   - ❑ fals, adevărat
   - ❑ fals, fals
5. Care dintre variante este adevărată, dacă la execuția comenzii

```
1 su rrazvan
```

   se obține un prompt de shell fără a se solicita autentificare:
   - ❑ comanda a fost lansată de utilizatorul `root`
   - ❑ comanda a fost lansată de utilizatorul `rrazvan`
   - ❑ comanda a fost lansată din directorul `/etc`
   - ❑ comanda a fost lansată din directorul `/root`
6. Care din următoarele NU reprezintă un sistem de fișiere?
   - ❑ FAT32
   - ❑ swap
   - ❑ ext3
   - ❑ NTFS
7. Care din următoarele NU este un tip de partiție?
   - ❑ extinsă
   - ❑ logică
   - ❑ primară
   - ❑ virtuală
8. Care din următoarele distribuții este derivată din Debian?
   - ❑ Ubuntu
   - ❑ Fedora
   - ❑ Gentoo
   - ❑ OpenSuSE
9. Care din următoarele NU se referă la CLI?
   - ❑ comandă
   - ❑ prompt
   - ❑ terminal
   - ❑ window manager

10. Care din următoarele comenzi NU este o comandă de ajutor pe un sistem Linux?

- ❑ man
- ❑ info
- ❑ apropos
- ❑ adduser

Capitolul 3

Gestiunea pachetelor și utilizatorilor

Users */nm./: collective term for those who use computers. Users are divided into three types: novice, intermediate and expert.*
Novice Users*: people who are afraid that simply pressing a key might break their computer.*
Intermediate Users*: people who don't know how to fix their computer after they've just pressed a key that broke it.*
Expert Users*: people who break other people's computers.*

The Jargon File

Ce se învață din acest capitol?

- Gestiunea utilizatorilor și grupurilor în Linux
- Fișiere folosite pentru gestiunea utilizatorilor
- Ce este un pachet
- Care sunt operațiile de bază asupra unui pachet în Linux
- Gestiunea pachetelor în Debian/Ubuntu: utilitarele apt și dpkg
- Personal Package Archive (PPA)

3.1 Gestiunea utilizatorilor

Gestiunea utilizatorilor se referă la adăugarea de noi utilizatori, ștergerea unui utilizator existent, modificarea informațiilor despre un utilizator și afișarea diverselor informații.

În sistemele Unix, informațiile despre utilizatori sunt reținute în fișierul `/etc/passwd`. Fiecare linie din acest fișier conține numele utilizatorului, identificatorului său, home-ul, shell-ul rulat în momentul autentificării și alte informații de descriere:

```
root@anaconda:~# cat /etc/passwd
andreir:x:1114:1026:Andrei Rizoiu:/home/students/andreir:/bin/bash
alexn:x:1115:1026:Alex Negrea:/home/students/alexn:/bin/bash
[...]
```

Din motive de securitate, hash-ul asociat parolei nu se găsește în fișierul `/etc/passwd`, ci în fișierul `/etc/shadow` care nu poate fi accesat de majoritatea utilizatorilor (vezi secțiunea 10.2.3):

```
root@anaconda:~# ls -l  /etc/shadow
-rw-r----- 1 root shadow 7068 2008-09-12 11:59 /etc/shadow
```

Pentru a afla informații despre un utilizator al sistemului se pot folosi comenzile **`id`** sau **`finger`**:

```
root@anaconda:~# id andreir
uid=1114(andreir) gid=1026(students) groups=1026(students),1037(rl)

root@anaconda:~# finger alexn
Login: alexn                            Name: Alex Negrea
Directory: /home/students/alexn         Shell: /bin/bash
Never logged in.
No mail.
No Plan.
```

Utilizatorul privilegiat într-un sistem Unix este utilizatorul **root** cu uid-ul `0` și home-ul în `/root`:

```
root@anaconda:~# head -1 /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

Utilizatorul `root` (de fapt utilizatorul cu uid-ul 0) are drepturi absolute în cadrul sistemului și poate rula orice comandă. Se recomandă folosirea unui cont neprivilegiat. Doar atunci când este nevoie se va folosi contul privilegiat. Schimbarea unui utilizator se realizează cu ajutorul comenzii **`su`** urmată de introducerea parolei pentru acel utilizator. Dacă utilizatorul inițial este `root`, nu se solicită introducerea parolei:

```
root@anaconda:~# head -1 /etc/passwd
root:x:0:0:root:/root:/bin/bash

root@anaconda:~# su - andreir
andreir@anaconda:~$ su - razvan
Password:
```

Sistemele Ubuntu dezactivează, de obicei, utilizatorul `root` și recomandă folosirea comenzii **`sudo`**. De obicei, un utilizator neprivilegiat care are drept de sudo va rula comanda **`sudo bash`** pentru a obține un shell cu drepturi privilegiate (vezi secțiunea 10.3.3).

Schimbarea parolei unui utilizator se realizează cu ajutorul comenzii **`passwd`**. Utilizatorul privilegiat poate schimba parola oricărui utilizator. Un utilizator neprivilegiat își poate schimba parola doar sieși:

```
anaconda:~# passwd guest
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

3.1.1 UID, GID

Fiecărui utilizator din sistem îi este asociat un identificator numit `UID` (User Identifier). UID este un număr întreg ce poate lua valori între 0 și 32767 sau 65535, funcție de distribuție. Valoarea 0 este rezervată pentru utilizatorul `root`.

Pentru a găsi identitatea unui utilizator sistemul de operare folosește UID, și nu numele utilizatorului. De aceea, dacă în fișierul `/etc/passwd` se înlocuiește UID-ul unui utilizator normal cu 0, acel utilizator va căpăta drepturi administrative egale cu ale utilizatorului `root`.

Pe sistemele de operare Linux utilizatorii pot fi organizați în grupuri. În acest fel se pot realiza diverse configurări administrative mai ușor, aplicându-se politici la nivelul întregului grup și nu per utilizator. Un grup poate conține mai mulți utilizatori. Un utilizator poate face parte din mai multe grupuri.

Pentru fiecare grup sistemul de operare asociază un identificator, numit GID (Group Identifier). Acest identificator este un număr întreg ce poate lua valori între 0 și 32767 (limita superioară poate să difere în funcție de distribuție). GID 0 este rezervat pentru grupul utilizatorului root. La fel ca la UID, sistemul de operare folosește GID pentru a identifica un anumit grup și nu numele grupului.

3.1.2 Adăugarea și ștergerea utilizatorilor

Adăugarea unui utilizator poate fi realizată doar de `root`, în două moduri. Primul dintre ele este utilizarea comenzii **useradd**. Utilizarea acestei comenzi este greoaie, motiv pentru care adăugarea unui utilizator se realizează în mod uzual cu ajutorul comenzii **adduser**. **adduser** este un script care apelează la rândul lui **useradd**.

Apelarea **adduser** se poate realiza cu următorii parametri:

```
1 adduser [options] [--home DIR] [--shell|-s SHELL] [--no-create-home
  ] [--uid ID] [--firstuid ID] [--lastuid ID] [--ingroup GROUP | --gid ID]
    [--disabled-password] [--disabled-login] [--gecos GECOS] [--
  add_extra_groups] user
```

Dacă **adduser** este apelat doar cu numele utilizatorului ce va fi creat, el va porni un asistent care va cere toate datele necesare creării contului:

```
1  root@asgard:/home/george# adduser uso
2  Adding user 'uso' ...
3
4  Adding new group 'uso' (1001) ...
5
6  Adding new user 'uso' (1001) with group 'uso' ...
7
8  Creating home directory '/home/uso' ...
9
10 Copying files from '/etc/skel' ...
11
12 Enter new UNIX password:
13 Retype new UNIX password:
14 passwd: password updated successfully
15 Changing the user information for uso
16 Enter the new value, or press ENTER for the default
```

```
17          Full Name []: uso
18          Room Number []: EG306
19          Work Phone []:
20          Home Phone []:
21          Other []:
22  Is the information correct? [y/N] y
23
24  root@asgard:/home/george#
```

Din motive de securitate caracterele tastate în dreptul parolei nu vor fi afișate.

La crearea unui utilizator este creat și un grup cu același nume la care este adăugat utilizatorul creat.

Ştergerea unui utilizator se poate realiza cu **`userdel`** sau cu **`deluser`** (**`deluser`** este un script care apelează la rândul lui **`userdel`**):

```
1  root@asgard:/home/george# deluser uso
2  Removing user 'uso' ...
3
4  Done.
```

3.1.3 Adăugarea și ștergerea unui grup de utilizatori

Adăugarea unui grup de utilizatori se poate realiza cu ajutorul comenzii **`groupadd`**, sau cu ajutorul comenzii **`addgroup`**. **`addgroup`** este un script care apelează la rândul lui **`groupadd`**.

```
1  root@asgard:/home/george# addgroup usogroup
2  Adding group 'usogroup' (GID 1002) ...
3
4  Done.
```

Ştergerea unui grup de utilizatori se realizează tot prin două comenzi, **`groupdel`** și **`delgroup`** (**`delgroup`** este un script care apelează **`groupdel`**).

```
1  root@asgard:/home/george# delgroup usogroup
2  Removing group 'usogroup' ...
3
4  Done.
```

3.1.4 Modificarea datelor unui utilizator

Modificarea datelor unui utilizator se realizează cu ajutorul comenzii **`usermod`**. Aceasta permite modificarea grupului din care face parte un utilizator, a directorului de bază (*home*), a numelui de utilizator, a parolei etc.

De exemplu, pentru modifica directorul de bază al utilizatorului `george` se folosește opțiunea `-d`:

```
1  uso@asgard:~$ cd ~
2
3  uso@asgard:~$ pwd
4  /home/ uso
5
```

```
uso@asgard:~$ sudo su

root@asgard:/home/uso# usermod -d /home/usodir/ uso

root@asgard:/home/uso# exit

exit

uso@asgard:~$ exit

uso@asgard:~$ pwd
/home/uso

uso@asgard:~$
```

În exemplul anterior a fost schimbat directorul curent în directorul de bază al utilizatorului `uso`, apoi a fost afișat directorul curent (pentru a vedea care este directorul de bază actual), a fost pornit un nou shell în care utilizatorul autentificat a fost `root`, a fost modificat directorul de bază al utilizatorului `uso` în `/home/usodir`, a fost închis shell-ul de `root`, apoi și utilizatorul `uso` a părăsit sistemul de operare. Schimbarea directorului de bază al unui utilizator se realizează doar după ce utilizatorul părăsește sistemul și se autentifică din nou:

```
uso@asgard:~$ pwd
/home/usodir/
```

Modificarea parolei unui utilizator se poate face și cu ajutorul comenzii **`passwd`**:

```
root@asgard:/home/uso# passwd uso
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Din motive de securitate caracterele tastate în dreptul parolei nu vor fi afișate.

Dacă nu este specificat niciun nume de utilizator va fi schimbată parola utilizatorului curent. Doar utilizatorul `root` poate schimba parola altui utilizator.

3.1.5 Adăugarea și ștergerea utilizatorilor

În sistemele Debian-based, adăugarea, respectiv ștergerea unui utilizator se realizează prin intermediul scripturilor adduser și deluser:

```
root@anaconda:~# adduser test
Adding user 'test' ...
Adding new group 'test' (1038) ...
Adding new user 'test' (1003) with group 'test' ...
Creating home directory '/home/test' ...
Copying files from '/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for test
Enter the new value, or press ENTER for the default
        Full Name []: Test User
        Room Number []:
```

```
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [y/N] y

root@anaconda:~# id test
uid=1003(test) gid=1038(test) groups=1038(test)

root@anaconda:~# deluser --remove-home test
Looking for files to backup/remove ...
Removing files ...
Removing user `test' ...
Done.
```

Avantajul și, în același timp, dezavantajul folosirii comenzii **adduser** este interactivitatea. Automatizarea sarcinilor presupune comenzi non-interactive. Pentru aceasta, se pot folosi comenzile **useradd**, **userdel** și **usermod**. **useradd**, respectiv **userdel** sunt folosite de scripturile **adduser** și **deluser**.

```
root@anaconda:~# useradd -m -d /home/test test

root@anaconda:~# id test
uid=1116(test) gid=1116(test) groups=1116(test)

root@anaconda:~# usermod -s /bin/sh test

root@anaconda:~# userdel -r test

root@anaconda:~# id test
id: test: No such user
```

3.2 Gestiunea pachetelor

> Un **pachet**, sau un pachet software, este o aplicație sau o componentă accesibilă în forma unei arhive care poate fi instalată de un sistem de gestiune a pachetelor (**PMS** – *Package Management System*).

De obicei, pachetele reprezintă programe precompilate care pot fi instalate ușor, spre deosebire de instalarea din surse care este mai anevoioasă (vezi secțiunea 14.6.1).

Lucrul cu pachete (instalare, dezinstalare, configurare) se realizează prin intermediul unui sistem de gestiune a pachetelor (precum **dpkg**, **rpm**, **pacman**). Un astfel de sistem conține, în afară de fișierele asociate programului și un set de metainformații precum versiunea pachetului, descrierea și dependențele acestuia. PMS-ul folosește aceste informații pentru a decide dacă se va realiza instalarea pachetului, upgrade-ul acestuia, instalarea dependențelor etc. O dependență între pachetele A și B înseamnă că instalarea pachetului A necesită instalarea pachetului B. La fel, dezinstalarea pachetului B va forța dezinstalarea pachetului A.

Majoritatea distribuțiilor GNU/Linux folosesc noțiunea de *repository* (depozit de pachete). Acesta este un URL care precizează locația diverselor pachete ale distribuției. Aceste depozite sunt precizate în fișiere de configurare specifice distribuției.

Aplicații front-end peste PMS pot interoga depozitele și pot descărca și instala noi pachete.

În lumea Linux există diverse formate de pachete, cele mai cunoscute fiind formatul **DEB**, specific distribuțiilor Debian-based și formatul **RPM** folosit de Fedora/RedHat, Mandriva, SuSE, etc. Fiecare format are propriul PMS. Utilitarul **`alien`**[1] permite conversia între diverse formate de pachete.

Gestiunea pachetelor DEB

Utilitarul de bază (PMS) pentru gestiunea pachetelor DEB este **`dpkg`**. Folosit efectiv pentru instalarea sau dezinstalarea unui pachet, acest utilitar oferă opțiuni pentru interogarea stării actuale a pachetelor sau a conținutul acestora. Printre opțiunile utile se numără:

- listarea conținutului unui pachet

```
root@anaconda:/tmp# dpkg -L coreutils
/.
/bin
/bin/mkdir
/bin/mv
/bin/true
/bin/mknod
/bin/sleep
/bin/touch
/bin/chgrp
/bin/uname
/bin/echo
/bin/sync
[...]
```

- afișarea pachetelor al căror nume se potrivește cu o expresie regulată

```
root@anaconda:/tmp# dpkg -l 'linux*'
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-
installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err:
uppercase=bad)
||/ Name           Version        Description
+++-==============-==============-====================================

un  linux          <none>         (no description available)
un  linux-doc-2.6. <none>         (no description available)
un  linux-gnu      <none>         (no description available)
un  linux-image    <none>         (no description available)
un  linux-image-2. <none>         (no description available)
ii  linux-image-2. 2.6.18+6etch3  Linux kernel 2.6 image on Ppro/
Cele
[...]
```

- căutarea pachetului ce conține un anumit fișier

```
root@anaconda:/tmp# dpkg -S /bin/ps
procps: /bin/ps
```

[1] http://kitenet.net/~joey/code/alien/

În mod obișnuit, însă, cea mai mare parte a acestor acțiuni vor fi realizate prin intermediul utilitarului **APT** (*Advanced Packaging Tool*). APT este un front-end peste **dpkg** și permite interogarea depozitelor de pachete configurate, verificarea dependențelor, descărcarea automată a pachetelor din repository, actualizarea acestora, upgrade-ul unei distribuții etc.

Fișierul de configurare a unui depozit DEB pentru folosirea cu **APT** este `/etc/apt/sources.list`:

```
root@anaconda:/tmp# cat /etc/apt/sources.list
[...]
deb http://ftp.lug.ro/debian etch main contrib non-free
deb-src http://ftp.lug.ro/debian etch main contrib non-free
```

Adăugarea unui nou depozit înseamnă adăugarea unei noi linii în fișierul de configurare.

Acțiunile care pot fi realizate cu ajutorul utilitarului **apt** sunt:

- actualizarea listei de pachete

```
root@anaconda:/tmp# apt-get update
Get: 1 http://ftp.lug.ro etch Release.gpg [386B]
Get: 2 http://ftp.lug.ro etch/updates Release.gpg [189B]
Hit http://ftp.lug.ro etch Release
Get: 3 http://www.backports.org etch-backports Release.gpg [189B]
Get: 4 http://ftp.lug.ro etch/updates Release [37.6kB]
Ign http://debian.pkgs.cpan.org unstable Release.gpg
Get: 5 http://www.backports.org etch-backports Release [43.7kB]
Ign http://ftp.lug.ro etch/main Packages/DiffIndex
Ign http://ftp.lug.ro etch/contrib Packages/DiffIndex
[...]
```

- căutarea de pachete

```
root@anaconda:/tmp# apt-cache search hevea
hevea - translates from LaTeX to HTML, info, or text
lyx - High Level Word Processor
hevea-doc - HeVeA documentation
```

- afișarea de informații despre un fișier

```
root@anaconda:/tmp# apt-cache show hevea
Package: hevea
Priority: optional
Section: tex
Installed-Size: 2125
Maintainer: Debian OCaml Maintainers <debian-ocaml-maint@lists.
debian.org>
Architecture: all
Version: 1.09-3
```

- instalarea unui pachet (și a dependențelor sale)

```
root@anaconda:/tmp# apt-get install apt-file
Reading package lists... Done
Building dependency tree... Done
The following extra packages will be installed:
  libapt-pkg-perl libconfig-file-perl
The following NEW packages will be installed
  apt-file libapt-pkg-perl libconfig-file-perl
0 upgraded, 3 newly installed, 0 to remove and 66 not upgraded.
```

```
Need to get 106kB of archives.
After unpacking 406kB of additional disk space will be used.
Do you want to continue [Y/n]? Y
```

- dezinstalarea unui pachet

```
anaconda:/tmp# apt-get remove --purge apt-file
Reading package lists... Done
Building dependency tree... Done
The following packages will be REMOVED
  apt-file*
```

- curățarea cache-ului local de pachete

```
root@anaconda:/tmp# apt-get clean
instalarea surselor unui pachet

root@anaconda:/tmp# apt-get source apt-file
Reading package lists... Done
Building dependency tree... Done
Need to get 17.7kB of source archives.
Get: 1 http://ftp.lug.ro etch/main apt-file 2.0.8.2 (dsc) [505B]
Get: 2 http://ftp.lug.ro etch/main apt-file 2.0.8.2 (tar) [17.2kB]
```

- listarea tuturor pachetelor din cache-ul local de pachete, împreună cu dependețele acestora

```
root@anaconda:/tmp# apt-cache dump
Using Versioning System: Standard .deb
Package: pipenightdreams
 Version: 0.10.0-13
     File: /var/lib/apt/lists/ro.archive.ubuntu.
com_ubuntu_dists_jaunty_universe_binary-i386_Packages
  Depends: libc6 2.6.1-1
  Depends: libgcc1 1:4.2.1
  Depends: libsdl-image1.2 1.2.5
  Depends: libsdl1.2debian 1.2.10-1
  Depends: libstdc++6 4.2.1
  Depends: pipenightdreams-data 0.10.0-13
 Description Language:
                 File: /var/lib/apt/lists/ro.archive.ubuntu.
com_ubuntu_dists_jaunty_universe_binary-i386_Packages
                  MD5: 7d042c60ae2f422df1ada8160fb80333
...
```

- listarea repository-urilor din care face parte un pachet, versiunea disponibilă, versiunile candidate etc.

```
root@anaconda:/tmp#  apt-cache policy firefox
firefox:
  Installed: 3.0.14+build2+nobinonly-0ubuntu0.9.04.1
  Candidate: 3.0.14+build2+nobinonly-0ubuntu0.9.04.1
  Version table:
 *** 3.0.14+build2+nobinonly-0ubuntu0.9.04.1 0
        500 http://ro.archive.ubuntu.com jaunty-updates/main
Packages
        500 http://security.ubuntu.com jaunty-security/main Packages
        100 /var/lib/dpkg/status
     3.0.8+nobinonly-0ubuntu3 0
        500 http://ro.archive.ubuntu.com jaunty/main Packages
```

PPA

Există și conceptul de **Personal Package Archives**, un sistem ce-i permite programatorului să-și încarce pachetele proprii pentru a fi incluse în update-urile ulterioare de pachete, fiind astfel mai ușor distribuite către utilizatorii finali. Astfel, acesta va putea instala programul scris de altcineva ca pe un pachet normal.

Totuși, pentru a preveni problemele de securitate, acest lucru nu poate fi făcut oarecum. În continuare vom enumera pașii necesari pentru a putea instala programe dintr-un PPA, numit *someone561* necesar pentru a instala o versiune mai recentă a unor compilatoare.

1. Vom crea un fișier nou cu un conținut și o locație ca mai jos pentru a oferi detalii cu privire la adresa de unde se vor descărca pachetele

```
1 mihai@keldon:/tmp# cat /etc/apt/sources.list.d/haskell.list
2 deb http://ppa.launchpad.net/someone561/ppa/ubuntu jaunty main
3 deb-src http://ppa.launchpad.net/someone561/ppa/ubuntu jaunty main
```

   (se va schimba jaunty cu numele versiunii distribuției folosite)

2. Pasul următor constă în autentificarea acestui PPA. Deși poate fi făcută și prin editarea unui fișier sau este opțională, acest lucru nu este recomandat. Se preferă folosirea următoarei comenzi:

```
1 mihai@keldon:/tmp# apt-key adv --recv-keys --keyserver keyserver.
  ubuntu.com E51D9310
```

   unde ultimul argument reprezintă un cod unic de autentificare al posesorului PPA-ului. Dacă nu dorim să realizăm autentificarea vom avea mereu mesaje de avertisment în cazul instalării unor pachete din aceste repository-uri.

3. Ultimul pas va consta în reactualizarea cache-ului local de pachete și instalarea pachetelor din acest PPA

```
1 mihai@keldon:/tmp# apt-get update
2 ...
3 mihai@keldon:/tmp# apt-get install ghc6 ...
4 ...
```

update-alternatives

În timpul rulării Linux, este un caz obișnuit să existe mai multe programe având aproximativ același scop. În special, compilatoarele și editoarele de text sunt exemple des întâlnite ale acestui caz. Prin urmare, deseori apare problema alegerii programului potrivit.

Această problemă este accentuată de faptul că există multe programe ce vor apela alte programe după un nume – de exemplu, multe programe permit utilizatorilor să editeze un fișier, fără a fi ele însele un editor. Ar trebui deci rulat un alt editor din sistem, dar care din ele? Un exemplu ar fi managementul taskurilor utilizând cron.

Prin urmare, pentru o clasă de aplicații care pot folosi o suită de alte programe, este nevoie de o modalitate de a configura variante alternative. O variantă simplă ar fi folosirea unor variabile de mediu (EDITOR de exemplu).

Varianta propusă de Debian este de a avea un set de comenzi standard, precum **`editor`**, **`www-browser`**, **`view`** etc. De fapt, toate acestea sunt link-uri simbolice spre comanda reală.

Pentru a seta aceste link-uri, se va folosi comanda **`update-alternatives`**. Acest sistem poate funcționa în două moduri:

- automatic mode – sistemul decide singur către ce program vor indica link-urile
- manual mode – administratorul sistemului va decide acest lucru

Implicit, modul este automatic. Link-urile simbolice sunt structurate în grupuri conținând prioritățile diverselor programe (editorul cu prioritate maximă este cel ales în cazul rulării **`editor`**).

Putem realiza următoarele:

- listarea informațiilor despre un grup

```
mihai@keldon:~# update-alternatives --display editor
editor - status is auto.
 link currently points to /usr/bin/vim.gnome
/usr/bin/vim.tiny - priority 10
 slave editor.ru.1.gz: /usr/share/man/ru/man1/vim.1.gz
 slave editor.pl.ISO8859-2.1.gz: /usr/share/man/pl.ISO8859-2/man1/
vim.1.gz
 slave editor.it.ISO8859-1.1.gz: /usr/share/man/it.ISO8859-1/man1/
vim.1.gz
 slave editor.1.gz: /usr/share/man/man1/vim.1.gz
 slave editor.pl.UTF-8.1.gz: /usr/share/man/pl.UTF-8/man1/vim.1.gz
 slave editor.it.1.gz: /usr/share/man/it/man1/vim.1.gz
 slave editor.fr.UTF-8.1.gz: /usr/share/man/fr.UTF-8/man1/vim.1.gz
 slave editor.fr.1.gz: /usr/share/man/fr/man1/vim.1.gz
 slave editor.it.UTF-8.1.gz: /usr/share/man/it.UTF-8/man1/vim.1.gz
 slave editor.pl.1.gz: /usr/share/man/pl/man1/vim.1.gz
 slave editor.fr.ISO8859-1.1.gz: /usr/share/man/fr.ISO8859-1/man1/
vim.1.gz
/bin/ed - priority -100
 slave editor.1.gz: /usr/share/man/man1/ed.1.gz
/bin/nano - priority 40
 slave editor.1.gz: /usr/share/man/man1/nano.1.gz
/usr/bin/vim.gnome - priority 60
 slave editor.ru.1.gz: /usr/share/man/ru/man1/vim.1.gz
 slave editor.pl.ISO8859-2.1.gz: /usr/share/man/pl.ISO8859-2/man1/
vim.1.gz
 slave editor.it.ISO8859-1.1.gz: /usr/share/man/it.ISO8859-1/man1/
vim.1.gz
 slave editor.1.gz: /usr/share/man/man1/vim.1.gz
 slave editor.pl.UTF-8.1.gz: /usr/share/man/pl.UTF-8/man1/vim.1.gz
 slave editor.it.1.gz: /usr/share/man/it/man1/vim.1.gz
 slave editor.fr.UTF-8.1.gz: /usr/share/man/fr.UTF-8/man1/vim.1.gz
 slave editor.fr.1.gz: /usr/share/man/fr/man1/vim.1.gz
 slave editor.it.UTF-8.1.gz: /usr/share/man/it.UTF-8/man1/vim.1.gz
 slave editor.pl.1.gz: /usr/share/man/pl/man1/vim.1.gz
 slave editor.fr.ISO8859-1.1.gz: /usr/share/man/fr.ISO8859-1/man1/
vim.1.gz
Current 'best' version is /usr/bin/vim.gnome.
```

- listarea tuturor țintelor din grup

```
mihai@keldon:~# update-alternatives --list editor
/usr/bin/vim.tiny
/bin/ed
/bin/nano
/usr/bin/vim.gnome
```

- schimbarea interactivă a unei opțiuni

```
root@keldon:/home/mihai# update-alternatives --config www-browser

There are 2 alternatives which provide 'www-browser'.

  Selection    Alternative
-----------------------------------------------
          1    /usr/bin/w3m
*+        2    /usr/bin/elinks

Press enter to keep the default[*], or type selection number:
```

- schimbarea unui link simbolic (neinteractiv)

```
root@keldon:/home/mihai# update-alternatives --set www-browser /usr/
bin/elinks
Using '/usr/bin/elinks' to provide 'www-browser'.
```

Pentru configurarea tuturor alternativelor se folosește **`update-alternatives --all`**

3.3 Studii de caz

3.3.1 Fișierele în care sunt stocate informații despre utilizatori

Informațiile despre grupuri și utilizatori sunt stocate în trei fișiere text:

- `/etc/passwd` – informații despre conturile utilizatorilor
- `/etc/shadow` – informații despre parolele utilizatorilor
- `/etc/group` – informații despre grupurile de utilizatori

Fișierul `/etc/passwd` conține câte o linie pentru fiecare utilizator. Această linie are șapte câmpuri despărțite prin caracterul ":". Câmpurile sunt (în ordinea în care apar în fișier):

- Nume de utilizator
- Parola. Un caracter x în această poziție arată faptul că parola este stocată în fișierul /etc/shadow.
- UID
- GID
- Informații legate de utilizator (Nume, Adresa, Telefon)
- Directorul de bază (directorul *home*)

- *Shell*-ul folosit

Un exemplu de fișier `/etc/passwd` este:

```
root@asgard:/home/george# cat /etc/passwd
[...]
sync:x:4:65534:sync:/bin:/bin/sync
george:x:1000:1000:George,,,:/home/george:/bin/bash
uso:x:1001:1001:uso,EG306,,:/home/uso:/bin/bash
[...]
```

Inițial, tot în fișierul `/etc/passwd` se rețineau și parolele, dar, pe măsură ce atacurile bazate pe dicționare au devenit fezabile, parolele au fost mutate într-un fișier cu permisiuni `600` numit `/etc/shadow`.

Fișierul `/etc/shadow` conține și el câte o linie pentru fiecare utilizator. Pe fiecare linie se găsesc opt câmpuri despărțite prin caracterul ":". Câmpurile sunt (în ordinea în care apar în fișier):

- Numele de utilizator
- Parola în format criptat. Dacă în loc de parolă se află un caracter "!", utilizatorul respectiv nu se poate autentifica în sistem
- Data la care a fost realizată ultima schimbare de parolă (data este notată în numărul de zile trecute de la 1 ianuarie 1970)
- Numărul minim de zile care pot trece între două schimbări de parolă
- Numărul maxim de zile pentru care parola este activă
- Cu câte zile înainte de expirarea parolei este anunțat utilizatorul să își shimbe parola
- Numărul de zile care trec după expirarea parolei până când contul va fi dezactivat
- Data la care contul va expira (data este notată în numărul de zile trecute de la 1 ianuarie 1970)

Criptarea parolei se face folosind o funcție de hash, în general MD5[1], dar, de preferat, SHA-2[2]. Aceste funcții sunt *one-way*, deci nu se poate afla parola simplu nici măcar folosind `/etc/shadow`. La fiecare autentificare, parola introdusă de utilizator este trecută prin algoritmul de hash și comparată cu valoarea din `/etc/shadow`.

Mai multe detalii despre criptarea parolei se poat afla din pagina de manual a comenzii **`crypt`**.

Un exemplu de fișier `/etc/shadow` este:

```
root@asgard:/home/georgedir/# cat /etc/shadow
root:!:13774:0:99999:7:::
daemon:*:13620:0:99999:7:::
bin:*:13620:0:99999:7:::
sys:*:13620:0:99999:7:::
sync:*:13620:0:99999:7:::
george:$1$LBNk03w4$4O2R/uMfCs/R9LOmH6zDG0:13774:0:99999:7:::
uso:$1$IPSj/ETo$1Uqe1Nrqm0r4LYdMihh6x1:13775:0:99999:7:::
```

[1] http://en.wikipedia.org/wiki/MD5
[2] http://en.wikipedia.org/wiki/SHA-2

```
9 [...]
```

Fișierul `/etc/group` conține câte o linie pentru fiecare grup. Această linie are patru câmpuri despărțite prin caracterul ":". Câmpurile sunt (în ordinea în care apar în fișier):

- Numele grupului
- Parola (câmp ce nu mai este folosit)
- GID
- Numele utilizatorilor ce fac parte din grup, în afară de utilizatorul care are același nume cu numele grupului

Un exemplu de fișier `/etc/group` este:

```
1 root@asgard:/home/georgedir/# cat /etc/group
2 root:x:0:
3 lp:x:7:cupsys
4 scanner:x:104:cupsys,hplip,george
5 george:x:1000:
6 uso:x:1001:
7 [...]
```

3.3.2 Actualizarea unui sistem Debian sau Ubuntu

Utilitarul **`apt`** oferă posibilitatea de a actualiza toate pachetele instalate pe un sistem care folosește pachete de tip `.deb`.

Primul pas în acest sens este asigurarea că lista locală de pachete este actualizată. Acest lucru se face folosind **`apt-get update`**. După această operație, utilizatorul are două alternative pentru instalarea celor mai noi pachete disponibile:

- **`apt-get upgrade`** actualizează toate pachetele care pot fi actualizate *fără a instala sau dezinstala alte pachete*. Este, din acest motiv, alternativa mai conservativă, pentru că nu introduce dependențe adiționale, și nici nu dezinstalează programe potențial utilizate;
- **`apt-get dist-upgrade`** este mai agresiv și instalează sau dezinstalează pachete după cum dependențele versiunilor noi cer. De asemenea, această comandă soluționează eventualele conflicte[1] într-un mod "inteligent", adică, în cazul unui conflict, va actualiza pachetul pe care îl consideră mai important.

În cele din urmă, o comandă utilă după un upgrade este **`apt-get autoremove`**. Aceasta va dezinstala pachetele care nu au fost instalate explicit de utilizator, dar nici nu mai sunt necesare niciunui pachet actual.

Cuvinte cheie

- utilizator
- uid
- grup
- gid

[1] Un conflict apare când două pachete cer explicit versiuni diferite dintr-un al treilea, ori când unul cere instalarea, iar celălalt dezinstalarea.

- root
- id
- finger
- /etc/passwd
- /etc/shadow
- /etc/group
- passwd
- su
- adduser, deluser
- useradd, userdel
- pachet
- Package Mangement System
- DEB, RPM
- /etc/apt/sources.list
- depozit (*repository*)
- apt
- dpkg
- dependență
- instalare, dezinstalare
- actualizare (*update*)
- upgrade

Întrebări

1. Care este comanda folosită pentru adăugarea non-interactivă a unui utilizator în sistem?
   - ❑ **`adduser`**
   - ❑ **`useradd`**
   - ❑ **`newuser`**
   - ❑ **`finger`**
2. De ce nu sunt ținute hash-urile parolelor în fișierul `/etc/passwd`?
   - ❑ `/etc/passwd` se poate afla pe o partiție nesigură.
   - ❑ permisiunile asupra `/etc/passwd` nu asigură securitate maximă.
   - ❑ întrebarea este o capcană, hash-urile parolelor chiar se află în `/etc/passwd`.
   - ❑ `/etc/passwd` nu poate fi modificat cu ajutorul comenzii **`passwd`**, de aceea este nevoie de un fișier cu permisiuni speciale.
3. Care dintre următoarele afirmații este adevărată despre sistemele de pachete?
   - ❑ `rpm` este bazat pe `apt`
   - ❑ `dpkg` este un wrapper peste `apt`
   - ❑ `apt` este un wrapper peste `dpkg`
   - ❑ sistemele Red Hat folosesc `dpkg`
4. Care este mecanismul prin care **`update-alternatives`** schimbă programele folosite pentru anumite task-uri?
   - ❑ legături simbolice

- ❑ legături hard
- ❑ instalare și dezinstalare on-demand a pachetelor
- ❑ script-uri wrapper care citesc configurația de fiecare dată când programul este rulat și lansează alternativa corespunzătoare

5. Care dintre următoarele informații NU este reținută în `/etc/passwd`?
   - ❑ directorul *home* al utilizatorului
   - ❑ interpretorul de comenzi (*shell*-ul)
   - ❑ grupurile din care face parte
   - ❑ numele utilizatorului
6. Un Personal Package Archive (PPA):
   - ❑ conține versiuni ale unor pachete diferite de ceea se găsește în repository-urile centrale
   - ❑ poate înlocui în întregime repository-urile centrale
   - ❑ este inerent nesigur și nu poate fi protejat
   - ❑ poate fi folosit de orice distribuție

Capitolul 4

Sisteme de fișiere

The Unix 'file system'. Sure it corrupts your data, but look how fast it is!

Ce se învață din acest capitol?

- Ce sunt și cum se utilizează sistemele de fișiere
- Tipuri de sisteme de fișiere
- Utilizarea căilor relative și absolute
- Lucrul cu fișiere (creare, copiere, mutare etc.)
- Descriptori de fișiere
- Redirectarea comenzilor în fișiere
- Arhivarea și dezarhivarea
- Drepturi de acces la fișiere

Cititorii familiarizați cu noțiunile de fișier, sisteme de fișiere și cu terminologia asociată pot porni direct de la secțiunea 4.3, continuând apoi cu analiza detaliată a sistemelor de fișiere.

4.1 Noțiuni introductive

4.1.1 Ce este un sistem de fișiere

Fișierul (*file*) este reprezentarea logică a unei informații sub forma unei înșiruiri de octeți. Fișierul poate fi considerat ca fiind versiunea electronică a documentului scris.

Directorul (*directory*) este o entitate în care se pot regăsi fișiere și/sau alte directoare. Acesta poate fi considerat versiunea electronică a dosarului.

În interfețele grafice, directorul este de obicei denumit folder.

Fișierele organizează informațiile pe mediile de stocare. Mediile de stocare pot fi considerate spații continue de octeți. În aceste spații, se pot regăsi mai multe fișiere, de dimensiuni variabile.

Sistemul de fișiere reprezintă modul de organizare a fișierelor pe un mediu de stocare pentru a le face mai ușor accesibile. Organizarea include atât partea logică (modul în care sunt adresate fișierele) cât și partea fizică (modul în care sunt stocate fișierele ca înșiruire de octeți).

Fiind o componentă a sistemului de operare, sistemul de fișiere menține numele și atributele fișierelor și permite stocarea lor într-o ierarhie de directoare numită și arbore de directoare.

Pentru ca utilizatorii să poată avea acces la fișiere, sistemul de operare oferă o interfață pentru lucrul cu sistemul de fișiere. Această interfață poate fi una textuală în mod text cum este interpretorul de comenzi, sau o interfață grafică.

Orice mediu de stocare poate fi utilizat pentru scrierea și citirea de fișiere dacă se cunoaște sistemul de fișiere utilizat pentru organizare.

Sistemele de operare folosesc fișierele pentru a organiza date, indiferent dacă aceste date sunt ale utilizatorului sau sunt generate pe moment de sistem.

Spre exemplu, în Linux și în Mac OS X, orice informație se găsește într-un fișier: datele se regăsesc în fișiere; directoarele sunt și ele fișiere, dar cu atributul de director; fiecare dispozitiv poate fi accesat printr-un fișier, inclusiv mouse-ul, memoria și placa video.

Intern, Windows folosește o schemă asemănătoare structurii de fișiere și directoare pentru a denumi dispozitivele, dar această structură nu se suprapune peste structura de fișiere precum în Linux și în Mac OS.

4.1.2 Ierarhia sistemului de fișiere

Sistemele de fișiere permit utilizatorului să organizeze datele într-un mod accesibil. Structura cel mai des întâlnită pentru organizarea fișierelor este arborele. Tabelele de mai jos prezintă structura ierarhică din sistemele de operare cele mai cunoscute.

În Tabela 4.1 se găsește o reprezentare grafică parțială pentru o ierarhie dintr-un sistem de fișiere în Linux. În rădăcină (`/`) se regăsesc directoarele `home`, `bin`, `usr` ..., în `home` se regăsesc directoarele `ubuntu` și `myuser` ș.a.m.d.

Comparativ cu Linux, structura în Windows este mult mai simplă pentru directoarele aflate imediat în rădăcină. În schimb, o mare parte din directoarele importante se ascund în directorul `Windows`.

Tabelul 4.1: Ierarhia într-un sistem de fișiere din mediul Unix

Director	Conținut
/	directorul rădăcină
/bin	comenzi esențiale necesare boot-ării, întreținerii și depanării sistemului
/boot	fișiere necesare boot-ării, precum imaginea kernel-ului
/dev	fișiere speciale utilizate pentru accesul direct la dispozitivele hardware sau logice ale sistemului
/etc	fișiere pentru configurarea sistemului, precum inittab, fstab și hosts
/home	fișierele fiecărui utilizator din sistem – datele unui utilizator se găsesc în /home/username
/media	subdirectoare în care se montează unitățile optice, floppy etc.
/mnt	subdirectoare în care se montează alte sisteme de fișiere
/opt	pachete de aplicații de dimensiuni mari, accesibile tuturor utilizatorilor
/proc	sistem virtual de fișiere din care se obțin informații despre sistem și aplicațiile care rulează la un moment dat
/root	directorul home al utilizatorului root
/sbin	comenzi de bază accesibile numai utilizatorului root
/tmp	fișiere temporare
/usr	aplicații pentru uzul normal al sistemului de operare – /usr/local conține aplicațiile instalate/compilate de utilizator
/var	fișiere al căror conținut se schimbă foarte des, precum log-uri, fișiere temporare, cache (date reutilizabile), spool (date neprocesate)

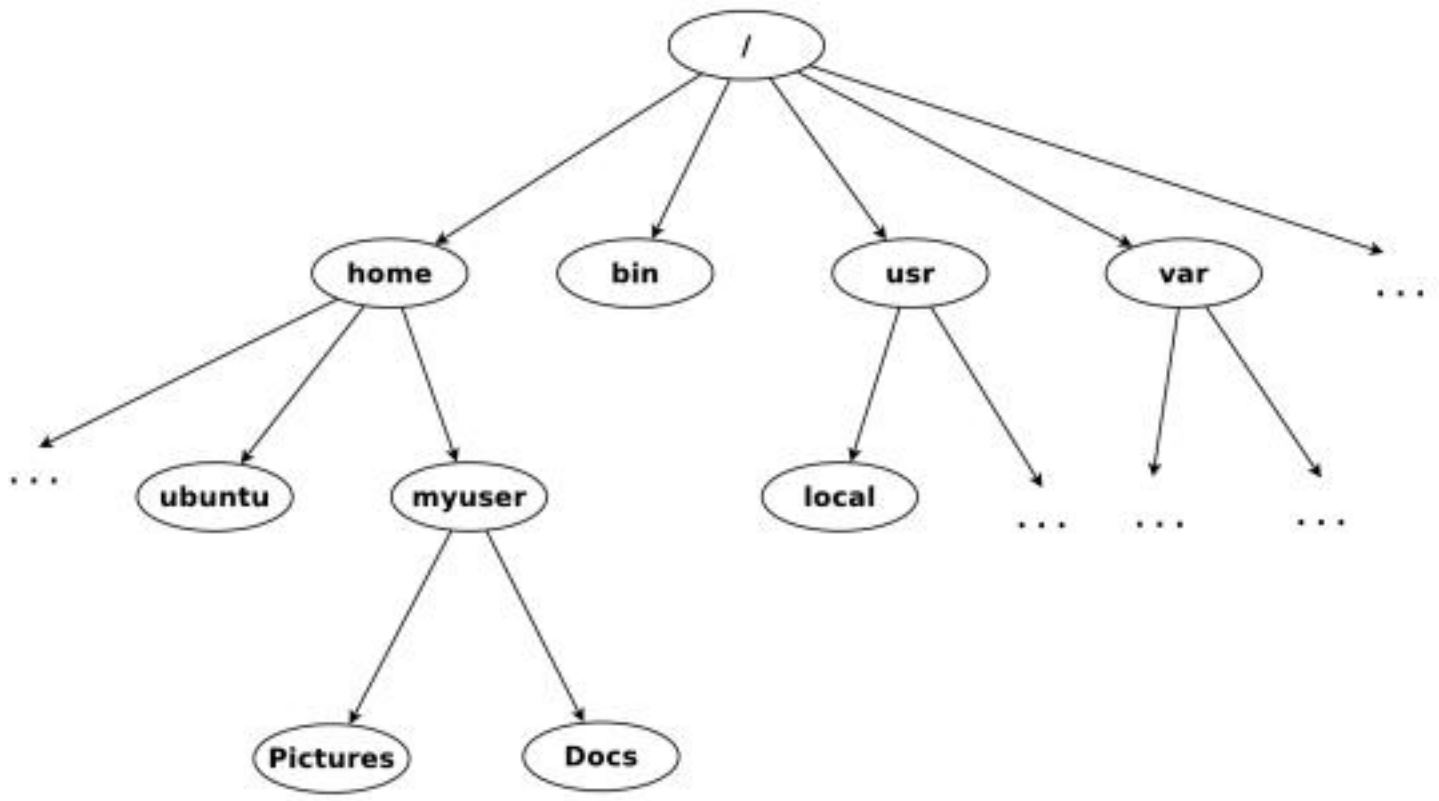


Figura 4.1: Ierarhia sistemului de fișiere în Linux

Tabelul 4.2: Ierarhia într-un sistem de fișiere din mediul Windows XP

Director	**Conținut**
`C:\`	directorul rădăcină
`C:\Windows`	Windows-ul și fișierele aferente
`C:\Documents and Settings`	configurările utilizatorilor și date specifice acestora
`C:\Program Files`	Aplicații
`C:\Windows\System32`	drivere și fișiere de configurare Windows
`C:\Documents and Settings\username\My Documents`	datele unui utilizator (aceasta este calea implicită, ea poate fi modificată)

Sistemul de fișiere folosit de Windows are o particularitate: există mai multe directoare rădăcină, câte unul pentru fiecare partiție sau disc. Spre exemplu, pot exista simultan următoarele directoare rădăcină:

- `A:` – floppy disk (de obicei literele `A:` și `B:` sunt rezervate de Windows pentru floppy disk-uri);
- `C:` – partiție de pe hard disk; dacă există mai multe partiții se asociază litere în ordine pentru fiecare dintre ele;
- `D:` – CD-ROM/DVD-ROM (următoarea literă disponibilă după ce s-au asociat litere partițiilor de pe hard disk(uri)).

După cum s-a precizat anterior, pe o partiție poate exista un singur sistem de fișiere la un moment dat. De obicei, sistemele de fișiere cu suport fizic se găsesc pe o singură partiție. Trebuie menționat faptul că Windows alocă literele după partiții, nu după sistemul de fișiere. Astfel, dacă sistemul de fișiere de pe o partiție se schimbă, acesta va avea asociată tot litera `C:`.

Tabelul 4.3 prezintă o analogie între căile importante din sistemele de operare majore existente în prezent.

4.1.3 Căi relative și căi absolute

Există două moduri prin care se poate accesa un fișier sau un director: folosind o cale absolută sau o cale relativă.

Calea absolută este calea care începe cu directorul rădăcină al sistemului de fișiere – în cazul Linux/Mac OS X este `/`; în cazul Windows este `C:`, `D:` etc.

Calea relativă este acea cale care indică spre un fișier pornind de la directorul curent.

Fiecare director conține două directoare speciale:

- `.` (punct) indică spre același director (directorul curent);
- `..` (punct, punct) indică spre directorul părinte.

Tabelul 4.3: Comparație între căile sistemelor de operare

Descriere	Windows	Linux	Mac OS X
radacină	`C:`	`/`	`/`
director home	`C:\Documents and Settings\username` (WinXP), `C:\Users\username` (Vista)	`/home/username`	`/Users/username`
aplicații	`C:\Program Files`	`/bin`; `/sbin`; `/usr/bin`; `/usr/sbin`; `/usr/local/bin` `/opt/*/bin`	`/Applications`; `/bin`; `/sbin`
configurări ale sistemului	Windows Registry	directoare specifice fiecărei aplicații, aflate în home-ul utilizatorului; `/etc`	`/Users/username/Library`; `/etc`

Astfel, dacă se dorește "urcarea" în ierarhia de fișiere, se folosește `..` care indică directorul părinte. Se poate urca mai mult în ierarhia de directoare înlănțuind câteva grupări de `..` delimitate prin separatorul de directoare.

În exemplul de mai jos:

- comanda **`pwd`** (*print working directory*) afișează directorul curent;
- comanda **`cd ../../..`** schimbă directorul curent în părintele-părintelui-părintelui directorului curent (s-ar putea spune "străbunicul directorului curent").

```
1 ubuntu@ubuntu:~/cs/uso$ pwd
2 /home/ubuntu/cs/uso
3
4 ubuntu@ubuntu:~/cs/uso$ cd ../../..
5
6 ubuntu@ubuntu:/home$ pwd
7 /home
```

`.` se folosește pentru a indica în mod explicit o cale care are ca punct de pornire directorul curent. `.` este utilizat frecvent pentru a scrie comenzi care execută script-uri/programe aflate în directorul curent.

Spre exemplu, comanda de mai jos execută fișierul `program_din_directorul_curent` – acest fișier trebuie să poată fi executat de utilizatorul curent.

```
1 ubuntu@ubuntu:~$ ./program_din_directorul_curent
```

Pentru mai multe exemple de utilizare a căilor relative și absolute se poate consulta secțiunea 4.4.

4.1.4 Variabila de mediu PATH

În mod normal, pentru execuția programelor uzuale, nu este necesară specificarea căii complete sau schimbarea directorului. Sistemele de operare utilizează variabile de mediu (environment variables) pentru a stoca diferite informații. `PATH` este o variabila de mediu care conține lista căilor uzuale unde se găsesc programe. Programele care se găsesc în oricare din căile din `PATH` pot fi executate fără a preciza calea către ele. Mai multe informații despre variabile de mediu și modificarea acestora se regăsesc în secțiunea 12.10.1.

Pentru a afișa variabila de mediu `PATH` într-un mediu Linux/Mac OS X se utilizează comanda:

```
1 ubuntu@ubuntu:~$ echo $PATH
2 /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

Căile din listă sunt despărțite prin separatorul `:` (două puncte).

Într-un mediu Windows, comanda este:

```
1 C:> echo %PATH%
2 C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem
```

Separatorul folosit în Windows este `;` (punct și virgulă).

4.2 Tipuri de fișiere

4.2.1 Terminologie

Sistemele de fișiere pun la dispoziție utilizatorilor diferite forme logice de organizare și acces la date, bineînțeles sub formă de fișiere.

În afară de directoare și fișiere, sistemul de fișiere pune la dispoziția utilizatorului și legături (link-uri).

Astfel, sistemele de fișiere moderne oferă posibilitatea utilizării mai multor tipuri de fișiere, prezentate în tabelul 4.4.

Tabelul 4.4: Tipuri de fișiere

Tipuri fișier	Denumire tip în engleză	Descriere
fișiere normale	regular files	unitate logică de acces la date
directoare	directories/folders	organizează fișiere și alte directoare
legături simbolice	symbolic links	legătura către un alt fișier

Există posibilitatea de a crea legături atât pentru fișiere cât și pentru directoare – în fond, ambele entități sunt fișiere (dar cu atribute diferite). Legăturile se utilizează la

scriere/citire la fel fișierele normale. Sistemul de operare împreună cu sistemul de fișiere se ocupă transparent de "traducerea" acestora în fișiere normale pentru aplicații.

Link-urile sunt create cu ajutorul comenzii **`ln`**. Mai jos sunt prezentate două exemple de folosire. În primul exemplu se creează un link al fișierului `/etc/apt/sources.list`, iar apoi la un fișier local precizându-se un nou nume pentru link.

```
ubuntu@ubuntu:~/tmp$ ls -l
total 1
-rw-r--r-- 1 ubuntu ubuntu  37 2007-09-04 10:59 hello.txt

ubuntu@ubuntu:~/tmp$ ln -s /etc/apt/sources.list

ubuntu@ubuntu:~/tmp$ ls -l
total 2
-rw-r--r-- 1 ubuntu ubuntu  37 2007-09-04 10:59 hello.txt
lrwxrwxrwx 1 ubuntu ubuntu  17 2007-09-04 10:59 sources.list -> /etc/apt/
sources.list

ubuntu@ubuntu:~/tmp$ ln -s hello.txt hello_link.txt

ubuntu@ubuntu:~/tmp$ ls -l
total 3
lrwxrwxrwx 1 ubuntu ubuntu   9 2007-09-04 11:00 hello_link.txt -> hello.
txt
-rw-r--r-- 1 ubuntu ubuntu  37 2007-09-04 10:59 hello.txt
lrwxrwxrwx 1 ubuntu ubuntu  17 2007-09-04 10:59 sources.list -> /etc/apt/
sources.list
```

4.2.2 Detecția tipului fișierelor

Pe sistemele Windows, detecția tipului fișierelor se bazează pe verificarea extensiei. În funcție de aceasta, sistemul de operare determină care aplicație este potrivită pentru a deschide un fișier.

Pe sistemele Linux, detecția tipului fișierelor este realizată prin intermediul comenzii **`file`**. Aceasta utilizează un fișier special de configurare în care sunt înscrise secvențe de octeți care se regăsesc în diferite tipuri de fișiere. **`file`** lucrează independent de extensia fișierului, precum este dovedit în exemplele următoare:

```
ubuntu@ubuntu:~$ file photo.jpg
photo.jpg: JPEG image data, JFIF standard 1.01

ubuntu@ubuntu:~$ mv photo.jpg fisier.txt

ubuntu@ubuntu:~$ file fisier.txt
photo.txt: JPEG image data, JFIF standard 1.01
```

4.3 Operații uzuale asupra fișierelor și directoarelor

4.3.1 Afișarea și schimbarea directorului curent

Comanda **pwd** afișează calea absolută către directorul curent. De asemenea, prompt-ul bash conține implicit directorul curent. Pentru a schimba directorul curent se folosește comanda **cd <cale>**. Calea poate să fie o cale absolută sau relativă.

Mai jos se găsește un exemplu de folosire a comenzilor **pwd** și **cd** – a se observa că la fiecare schimbare de director se modifică și prompt-ul bash. Ierarhia de directoare prin care se navighează este cea prezentată anterior.

```
ubuntu@ubuntu:~$ pwd
/home/ubuntu

ubuntu@ubuntu:~$ cd ..

ubuntu@ubuntu:/home$ pwd
/home

ubuntu@ubuntu:/home$ cd ../usr/bin

ubuntu@ubuntu:/usr/bin$ pwd
/usr/bin

ubuntu@ubuntu:/usr/bin$ cd .

ubuntu@ubuntu:/usr/bin$ cd /

ubuntu@ubuntu:/$ cd

ubuntu@ubuntu:~$ pwd
/home/ubuntu

ubuntu@ubuntu:~$ cd /usr/bin

ubuntu@ubuntu:/usr/bin$ cd /home

ubuntu@ubuntu:/home$ cd -
/usr/bin

ubuntu@ubuntu:/usr/bin$ cd ~

ubuntu@ubuntu:~$ pwd
/home/ubuntu

ubuntu@ubuntu:~$ cd ./././.

ubuntu@ubuntu:~$ cd ../..

ubuntu@ubuntu:/$ pwd
/

ubuntu@ubuntu:/$
```

Observații:

- **cd ..** schimbă directorul în directorul părinte;

- . (punct) este o referință la directorul curent, deci **cd .** nu modifică directorul;
- ~ este echivalentul directorului home pentru utilizatorul curent (**cd** ~);
- **cd** apelat fără parametri schimbă directorul în directorul home;
- **cd -** schimbă directorul în directorul anterior.

4.3.2 Afișarea conținutului fișierelor

Conținutul unui fișier poate fi afișat pe ecran prin intermediul comenzii **cat**. Sintaxa acestei comenzi este **cat <nume fișier>**. Un exemplu de utilizare este:

```
1 ubuntu@ubuntu:~$ cat /etc/resolv.conf
2 # Generated by dhcpcd for interface eth0
3 search localdomain
4 nameserver 192.168.0.1
```

Această comandă afișează tot fișierul, inclusiv dacă acesta este mai mare de un ecran. Pentru a putea naviga prin output-ul unei comenzi (în cazul acesta, al comenzii **cat**), se pot folosi comenzile **more** sau **less**.

more permite navigarea doar intr-o singură direcție (de la început către sfârșitul fișierului) și câte un ecran odată, pe când **less** permite navigarea în ambele direcții, câte o linie, la fel ca un editor. **more** și **less** poartă numele de **paginatoare** (*pagere*). Interfața **less** este foarte asemănătoare cu cea a editorului Vim (vezi secțiunea 14.3).

```
1 ubuntu@ubuntu:~$ cat /etc/X11/xorg.conf | more
2
3 ubuntu@ubuntu:~$ cat /etc/X11/xorg.conf | less
```

4.3.3 Listarea conținutului unui director

Cea mai frecventă operație care se execută asupra directoarelor este listarea. Comanda utilizată este **ls**. În această secțiune, prin fișier înțelegem orice tip de fișier (inclusiv tipul "director"), mai puțin în cazurile când este menționat explicit.

Sintaxa comenzii pentru listare este **ls [opțiuni] [<cale>]**. În momentul în care lipsește calea, se realizează listarea conținutului directorului curent.

Opțiunile cele mai des folosite cu aceasta comandă sunt:

- `-l` afișează informații detaliate despre fiecare fișier/director (data modificare, dimensiune, utilizator, grup, drepturi de acces);
- `-a` afișează și fișierele care încep cu . (caracterul punct); în Unix, aceste fișiere sunt considerate de majoritatea interfețelor cu utilizatorul ca fiind fișiere ascunse;
- `-h` afișează dimensiunea fișierelor în format human-readable, respectiv dimensiunea în octeți este înlocuită cu dimensiunea în Kiloocteți/Megaocteți/Gigaocteți dacă depășește un anumit ordin de mărime.

Formatul utilizat la afișarea detaliată conține mai multe informații despre acel fișier:

```
ubuntu@ubuntu:~$ ls -l /home/myuser/myapp
-rwxr----- 1 myuser users 1176348 2007-03-03 20:28 /home/myuser/myapp
```

Semnificația coloanelor afișate este:

- primul caracter reprezintă tipul fișierului:
  - `-` = fișier normal
  - `d` = director
  - `p` = pipe
  - `b` = dispozitiv bloc
  - `c` = dispozitiv caracter
  - `l` = legătură simbolică
- urmează 3 grupuri de câte 3 caractere (`rwx`) care reprezintă drepturile de acces pentru utilizatorul `myuser` (care este deținătorul fișierului), drepturile de acces pentru utilizatorii care fac parte din grupul `users` (care deține fișierul), drepturile de acces pentru ceilați utilizatori; mai multe detalii despre drepturile de acces se găsesc în secțiunea 4.5;
- numărul de link-uri către fișier – `1`;
- utilizatorul ce deține fișierul - `myuser`;
- grupul de care aparține fișierul – `users`;
- dimensiunea fișierului – `1176348`;
- data fișierului – `2007-03-03`;
- ora fișierului – `20:28`;
- numele fișierului – `/home/myuser/myapp`.

Exemplele următoare prezintă câteva utilizări mai întâlnite pentru comanda **`ls`** care combină parametrii prezentați anterior:

```
ubuntu@ubuntu:~$ ls
Desktop

ubuntu@ubuntu:~$ ls ~
Desktop

ubuntu@ubuntu:~$ ls -l ~
total 0
drwxr-xr-x 2 ubuntu ubuntu 100 2007-09-03 13:15 Desktop

ubuntu@ubuntu:~$ ls -al
total 56
drwxr-xr-x 7 ubuntu ubuntu  580 2007-09-03 19:58 .
drwxr-xr-x 3 root   root     60 2007-09-03 06:14 ..
-rw------- 1 ubuntu ubuntu   34 2007-09-03 19:58 .bash_history
-rw-r--r-- 1 ubuntu ubuntu 2346 2007-09-03 06:14 .bashrc
drwxr-xr-x 2 ubuntu ubuntu  100 2007-09-03 13:15 Desktop
-rw-r--r-- 1 ubuntu ubuntu  566 2007-09-03 06:14 .profile
-rw------- 1 ubuntu ubuntu 9524 2007-09-03 17:40 .xsession-errors
```

```
20
21 ubuntu@ubuntu:~$ ls -alh
22 total 56K
23 drwxr-xr-x 7 ubuntu ubuntu  580 2007-09-03 19:58 .
24 drwxr-xr-x 3 root   root     60 2007-09-03 06:14 ..
25 -rw------- 1 ubuntu ubuntu   34 2007-09-03 19:58 .bash_history
26 -rw-r--r-- 1 ubuntu ubuntu 2.3K 2007-09-03 06:14 .bashrc
27 drwxr-xr-x 2 ubuntu ubuntu  100 2007-09-03 13:15 Desktop
28 -rw-r--r-- 1 ubuntu ubuntu  566 2007-09-03 06:14 .profile
29 -rw------- 1 ubuntu ubuntu 9.5K 2007-09-03 17:40 .xsession-errors
30
31 ubuntu@ubuntu:~$ ls ../../
32 bin    dev   initrd      media  proc  sbin  tmp  vmlinuz
33 boot   etc   initrd.img  mnt    rofs  srv   usr
34 cdrom  home  lib         opt    root  sys   var
35
36 ubuntu@ubuntu:~$ ls -a /
37 .    boot   etc     initrd.img  mnt   rofs  srv  usr
38 ..   cdrom  home    lib         opt   root  sys  var
39 bin  dev    initrd  media       proc  sbin  tmp  vmlinuz
```

Observații:

- dacă se dorește afișarea conținutului directorului curent, nu este necesară scrierea căii ca parametru pentru comanda **`ls`**;
- **`ls -al`** afișează lista detaliată a fișierelor din directorul home al utilizatorului curent, inclusiv fișierele ascunse (a se observă că și `.` și `..` sunt afișate;
- **`ls -alh`** afișează lista detaliată a fișierelor din directorul home al utilizatorului curent, utilizând formatul *human-readable* pentru afișarea dimensiunii fișierelor (a se observa dimensiunea fișierului `.xsession-errors`);
- **`ls ../..`** prezintă utilizarea lui **`ls`** cu parametru dat sub formă de cale relativă (directorul afișat este `/`);
- **`ls -a /`** prezintă fișierele ascunse ale directorului `/` – se observă că și acest director conține directoarele standard `.` și `...`

Folosind opțiunea `-R` se poate afișa arborele de directoare și fișiere care are ca rădăcină directorul specificat ca argument:

```
1 ubuntu@ubuntu:~$ ls -R dir1
2 dir1:
3    dir2  fisier1  fisier2
4 dir1/dir2:
5    fisier3  fisier4
```

Pentru mai multe opțiuni se poate consulta **`ls --help`** sau **`man ls`**.

4.3.4 Crearea fișierelor/directoarelor

Pentru a crea orice entitate pe un sistem de fișiere, se utilizează o serie de comenzi, prezentate în tabelul 4.5.

Primul mod de creare a unui fișier este utilizând comand comanda **`touch`**. A doua metodă se bazează pe o funcționalitate bash numită redirectare în fișiere (vezi

Tabelul 4.5: Comenzi pentru crearea fișierelor

Entitate	Comandă
Fișier normal	`touch <nume_fișier>`, `> <nume_fișier>`
Director	`mkdir <nume_director>`
Legături (link-uri)	`ln -s <destinatie> [<nume_legătură>]`
Pipe-uri cu nume	`mkfifo <nume_pipe>`

secțiunea 4.4). Pe scurt, ceea ce realizează comanda a 2-a este redirectarea ieșirii unei comenzi (nule) către un fișier, creându-se astfel un fișier gol.

```
ubuntu@ubuntu:~$ touch fisier1

ubuntu@ubuntu:~$ > fisier2

ubuntu@ubuntu:~$ mkdir dir1

ubuntu@ubuntu:~$ mkfifo fifo1

ubuntu@ubuntu:~$ ls -l
total 0
drwxr-xr-x 2 mircea users 40 2007-09-20 10:26 dir1
prw-r--r-- 1 mircea users  0 2007-09-20 10:30 fifo1
-rw-r--r-- 1 mircea users  0 2007-09-20 10:26 fisier1
-rw-r--r-- 1 mircea users  0 2007-09-20 10:26 fisier2
```

O altă întrebuințare a comenzii **`touch`** este aceea a actualizării datei ultimei modificări și a datei ultimei accesări. Folosind opțiunea `-m` se va actualiza doar data ultimei modificări iar folosind opțiunea `-a` se va actualiza doar data ultimei accesări. Folosind comanda **`touch`** fără aceste opțiuni va duce la modificarea ambelor date. Optiunea `-r` poate fi folosită pentru a prelua informațiile legate de timp de la alt fișier.

În exemplul de mai jos, se observă modificarea timpului de acces al fișierului de la valoarea `17:03` la `17:11`.

```
ubuntu@ubuntu:~$ ls -al
-rw-r--r--  1 root     root       15 2009-08-21 17:03 fisier1

ubuntu@ubuntu:~$ touch fisier1

ubuntu@ubuntu:~$ ls -al
-rw-r--r--  1 root     root       15 2009-08-21 17:11 fisier1
```

Exemple privind crearea de link-uri sunt prezentate în secțiunea 4.2.1.

4.3.5 Copiere/mutare/redenumire/ștergere

Copierea și mutarea sunt operații care necesită 2 parametri:

- sursă de unde se copiază/mută;
- destinația unde se copiază/mută.

Copierea

Copierea unui fișier sau director se realizează cu ajutorul comenzii **cp**. Sintaxa comenzii este **cp [opțiuni] <sursa> <destinatie>**. Opțiunile cele mai folosite pentru copiere sunt:

- `-R` – copiere recursivă (copiază și copiii directoarelor, presupunând că există așa ceva în sursă);
- `-p` – copiere cu păstrare a tuturor atributelor (permisiuni, dată);
- `-u` – copiază doar dacă fișierul sursă este mai nou decât fișierul destinație sau dacă fișierul destinație lipsește.

```
1 ubuntu@ubuntu:~$ touch f1.txt
2
3 ubuntu@ubuntu:~$ ls
4 f1.txt
5
6 ubuntu@ubuntu:~$ cp f1.txt f2.txt
7
8 ubuntu@ubuntu:~$ ls
9 f1.txt  f2.txt
```

Mutarea

Mutarea unui fișier sau director se realizează cu ajutorul comenzii **mv**. Sintaxa comenzii este **mv [opțiuni] <sursa> <destinatie>**. Implicit, mutarea este recursivă și păstrează atributele fișierelor. În cazul în care sursa și destinația se găsesc pe aceeași partiție, la o mutare, se schimbă doar părintele fișierului sau directorului care se mută. O operație de mutare este, astfel, mai puțin costisitoare decât o operație de copiere.

```
1  ubuntu@ubuntu:~$ pwd
2  /tmp/q
3
4  ubuntu@ubuntu:~$ ls
5  f1.txt  f2.txt
6
7  ubuntu@ubuntu:~$ mv f1.txt /tmp/r
8
9  ubuntu@ubuntu:~$ ls
10 f2.txt
11
12 ubuntu@ubuntu:~$ ls /tmp/r/
13 f1.txt
```

Redenumirea

Redenumirea este, de fapt, o mutare și se realizează cu ajutorul comenzii **mv**, în următoarele conditii:

- în cazul în care sursa **este un fișier** și destinația **este un fișier**, se realizează copierea/mutarea fișierului cu schimbarea numelui sursei;

- în cazul în care sursa **este un director** și destinația **nu există ca director**, se realizeaza copierea/mutarea directorului cu schimbarea numelui.

```
ubuntu@ubuntu:~$ ls
f2.txt

ubuntu@ubuntu:~$ mv f2.txt f3.txt

ubuntu@ubuntu:~$ ls
f3.txt
```

Stergerea fișierelor/directoarelor

În Linux, comanda cea mai utilizata pentru ștergerea fișierelor și directoarelor este **`rm`**. Sintaxa comenzii este **`rm [opțiuni] <cale>`**.

Una dintre cele mai folosite opțiuni este `-r/-R`, utilizată pentru a șterge recursiv un director. Aceasta opțiune trebuie folosită cu atenție pentru ca poate avea rezultate negative dacă directorul care este șters conține informații utile. O eroare celebră este utilizarea comenzii **`rm -rf /`**, care șterge recursiv totul începând cu directorul rădăcină, forțând ștergerile (datorită opțiunii `-f`).

Pentru ștergerea directoarelor goale se poate folosi și `rmdir`.

4.3.6 Arhivarea fișierelor și dezarhivarea

> **Arhivarea** este procedeul prin care mai multe fișiere și directoare sunt adunate la un loc (într-un singur fișier), realizând eventual și o reducere a dimensiunii prin eliminarea datelor care se repetă și înlocuirea lor cu o serie de codificări.

În general, în cadrul procesului de arhivare se creează un dicționar cu secvențe de octeți frecvente și o codificare a lor mai scurtă (pe mai puțini octeți). Arhiva conține atât dicționarul cât și conținutul fișierelor dar, în loc de secvențele frecvente lungi, se regăsesc referințe către dicționar. Dacă fișierele conțin multe date care se repetă, dimensiunea unei arhive scade considerabil.

După cum se poate identifica și din descrierea anterioară, există două etape mai importante în ceea ce privește manevrarea fișierelor:

- concatenarea (lipirea) fișierelor într-un fișier mai mare, din care să se poată extrage toate fișierele și informațiile despre ele;
- compresia fișierului mare, astfel încât să se reducă dimensiunea lui dar fără a se pierde din informații.

În Linux există utilitare de arhivare care se ocupă ori de una dintre etape ori de ambele etape.

Comanda **`tar`** se ocupă de prima etapă. Numele utilitarului este o abreviere a *tape archive*, întrucât, la începutul utilizării sale, rezultatul comenzii era transferat pe benzi magnetice.

Un fișier `.tar` conține fișiere necomprimate împreună cu informații despre modul de extragere al acestora (spre exemplu: de unde până unde se găsește un fișier în cadrul arhivei). Din acest motiv, un fișier `.tar` este de obicei mai mare decât suma tuturor dimensiunilor fișierelor ce sunt incluse în el.

Pentru a crea o arhiva `.tar`, se utilizează comanda:

```
1 ubuntu@ubuntu:~$ tar cvf nume_arhiva.tar <cale>
```

cu parametrii:

- `c` creează arhiva;
- `v` afișează ce anume se arhivează;
- `f nume_arhiva.tar` specifică numele arhivei;
- `<cale>` precizează directorul/directoarele/fișiere care vor fi arhivate.

Pentru a dezarhiva o arhivă `.tar`, se utilizează:

```
1 ubuntu@ubuntu:~$ tar xvf nume_arhiva.tar [-C <cale_destinatie>]
```

unde:

- `x` dezarhivează (*eXtract*);
- `v` afișează ce anume se dezarhivează;
- `f nume_arhiva.tar` precizează numele arhivei care se dezarhivează;
- `-C <cale_destinatie>` specifică, opțional, locul unde se realizează dezarhivarea.

`f` și `nume_arhiva.tar` se consideră un singur parametru; din acest motiv, de fiecare dată când se folosește opțiunea `f` pentru a indica un fișier arhivă, aceasta trebuie să apară ultima în lista de opțiuni, fiind urmat imediat de numele fișierului arhivă.

Pentru a realiza compresia unui fișier, două utilitare sunt folosite preponderent în lumea Unix:

- **`gzip`**, mai rapid dar cu o rată de compresie mai mică;
- **`bzip2`**, mai lent dar cu o rată de compresie mai mare.

Comanda **`tar`** poate utiliza direct unul dintre programele de comprimare menționate anterior folosind parametrul `z` pentru **`gzip`** sau parametrul `j` pentru **`bzip2`**, ca în tabelul 4.7

Tabelul 4.6: Comprimarea și decomprimarea folosind gzip și bzip2

	Comprimare	**Decomprimare**
gzip	**`tar czvf fisier.tar.gz dir/`**	**`tar xzvf fisier.tar.gz`**
bzip2	**`tar cjvf fisier.tar.bz2 dir/`**	**`tar xjvf fisier.tar.bz2`**

Pe lângă opțiunile de compresie/arhivare se mai pot folosi și alte opțiuni. Printre cele mai utile se numără opțiunea `--preserve`, care impune păstrarea drepturilor de acces în momentul arhivării/dezarhivării.

Câteva exemple de utilizare a comenzii **tar** pentru comprimare și decomprimare, în conjuncție cu **bzip2** și **gzip** sunt prezentate mai jos:

```
ubuntu@ubuntu:/tmp/q$ ls
f1.txt  f2.txt  f3.txt

ubuntu@ubuntu:/tmp/q$ tar cvf arhiva.tar f*.txt
f1.txt
f2.txt
f3.txt

ubuntu@ubuntu:/tmp/q$ file arhiva.tar
arhiva.tar: POSIX tar archive (GNU)

ubuntu@ubuntu:/tmp/q$ gzip arhiva.tar

ubuntu@ubuntu:/tmp/q$ ls
arhiva.tar.gz  f1.txt  f2.txt  f3.txt

ubuntu@ubuntu:/tmp/q$ file arhiva.tar.gz
arhiva.tar.gz: gzip compressed data, was "arhiva.tar", from Unix, last
modified: Thu Sep 20 11:37:30 2007

ubuntu@ubuntu:/tmp/q$ ls
arhiva.tar.gz  f1.txt  f2.txt  f3.txt

ubuntu@ubuntu:/tmp/q$ mv arhiva.tar.gz fisier.txt

ubuntu@ubuntu:/tmp/q$ file fisier.txt
fisier.txt: gzip compressed data, was "arhiva.tar", from Unix, last
modified: Thu Sep 20 11:37:30 2007

ubuntu@ubuntu:/tmp/q$ ls
f1.txt  f2.txt  f3.txt

ubuntu@ubuntu:/tmp/q$ tar czvf new.tar.gz f*
f1.txt
f2.txt
f3.txt

ubuntu@ubuntu:/tmp/q$ ls
f1.txt  f2.txt  f3.txt  new.tar.gz

ubuntu@ubuntu:/tmp/q$ file new.tar.gz
new.tar.gz: gzip compressed data, from Unix, last modified: Thu Sep 20
11:40:02 2007

ubuntu@ubuntu:/tmp/q$ mkdir tmp

ubuntu@ubuntu:/tmp/q$ mv new.tar.gz tmp/

ubuntu@ubuntu:/tmp/q$ cd tmp/

ubuntu@ubuntu:/tmp/q/tmp$ tar xvf new.tar.gz
f1.txt
f2.txt
f3.txt
ubuntu@ubuntu:/tmp/q/tmp$ ls
f1.txt  f2.txt  f3.txt  new.tar.gz
```

4.3.7 Backup

Backup-ul este utilizat pentru a păstra într-un loc separat o copie a datelor. Această copie se utilizează pentru a reface datele în cazul în care suportul original nu mai poate fi folosit din orice motiv. Este una dintre cele mai utile acțiuni asupra datelor pe care toată lumea știe că ar fi bine să o facă dar nu o face. Câteva metode de backup au fost deja prezentate. Tabelul următor prezintă câteva metode de backup și situațiile când sunt ele potrivite.

Tabelul 4.7: Comprimarea și decomprimarea folosind gzip și bzip2

Metodă	Descriere
`tar+gzip/bzip2`	Metodă foarte simplu de aplicat. Devine greu de folosit pentru dimensiuni mari de date. Permite comprimarea datelor.
`dd`	Metoda simplu de folosit și independentă de sistemul de fișiere. Permite păstrarea intactă a structurii sistemului de fișiere. Inflexibilă când vine vorba de recuperarea datelor. Utilă pentru cantități mari de date.
`rsync`[1]	E asemănătoare comenzii `cp` dar la care s-a adăugat suport de sincronizare intre mai multe sisteme de calcul. Permite replicarea structurii de fișiere (inclusiv permisiuni) între 2 sisteme de calcul.
`rdiff-backup`[2]	Este un wrapper peste `rsync`. Adaugă suport pentru backup-uri incrementale. La un moment dat se realizează un backup complet pentru un director (asemănător rsync-ului). Backup-urile incrementale salvează doar modificările ce s-au făcut de la ultimul backup până în prezent, indiferent de tipul backup-ului. În acest fel se poate reveni la orice stare anterioara, în măsura în care s-a realizat cel puțin un backup incremental la acea stare.

4.4 Redirectări de comenzi

4.4.1 Descriptorii de fișier

Fiecare program în execuție utilizează 3 fișiere speciale pentru a interacționa cu utilizatorul:

- **stdin** (*standard input*) – fișier care reprezintă locul de unde programul își citește datele de intrare (de obicei tastatura);
- **stdout** (*standard output*) – fișier care reprezintă locul unde programul scrie datele de ieșire (de obicei acesta este consola curentă);
- **stderr** (*standard error*) – fișier care reprezintă locul unde programul scrie mesajele de eroare (de obicei acesta este consola curentă).

În C/C++, există 3 variabile de tipul `FILE *` cu numele `stdin`, `stdout` și `stderr` care sunt inițializate și disponibile în orice moment. Ele îndeplinesc rolurile descrise anterior și pot fi folosite ca pe orice alta variabila de tipul `FILE *`.

Fiecare fișier deschis de o aplicație Linux are asociat un indice. Acest indice se numește descriptor de fișier. Fișierele descrise anterior au descriptorii de fișier următori:

- stdin – 0
- stdout – 1
- stderr – 2

Orice fișier deschis de aplicație va avea un descriptor mai mare sau egal cu 3.

4.4.2 Redirectări

Există situații când, pentru o aplicație, utilizatorul dorește să modifice intrarea sau ieșirea – spre exemplu, utilizatorul dorește ca ieșirea unui program să se facă într-un fișier și nu în consola curentă, sau intrarea unui program să fie un fișier și nu tastatura. Acest lucru se poate face doar lucrând la nivelul descriptorilor.

Cazurile întâlnite sunt prezentate în Tabela 4.8.

Tabelul 4.8: Comenzi de redirectare

sursă	**destinație**	**exemplu comandă**
intrare (stdin)	fișier	`./program < f_in`
ieșire (stdout)	fișier	`./program > f_out`
eroare (stderr)	fișier	`./program 2> f_err`
eroare (stderr)	ieșire (stdout)	`./program 2>&1`
eroare și ieșire	fișier	`./program 2>&1 > f_out_err` sau `./program &> f_out_err`

Se observă că pentru a redirecta ieșirea/intrarea se folosesc semnele `>` (*mai mare*) pentru ieșire (către exteriorul programului) și `<` (*mai mic*) pentru intrare (către program). Pentru redirectarea ieșirii de erori se indică descriptorul de fișier `2`. Pentru a redirecta către ieșirea standard (*stdout*), se folosește sintaxa `&1`.

Pentru redirectarea ieșirii de erori și iesirii standard către un fișier, se redirectează întâi ieșirea de erori către ieșirea standard folosind `2>&1` dupa care se redirectează ieșirea standard către un fișier `>fișier_iesire_și_erori`. **Bash** oferă o sintaxă simplificată în cadrul operatorului `&>`.

Ca exemplu de redirectare a intrării, în comanda anterioară se trimite conținutul fișierului `continut_mail` ca intrare pentru comanda **`mail`**. Se va trimite un mesaj utilizatorului **`user1`** cu informațiile din fișier.

```
1 ubuntu@ubuntu:~$ mail user1 < continut_mail
```

În exemplul de mai jos se scrie lista de fișiere și directoare din directorul curent în fișierul `listare`.

```
ubuntu@ubuntu:~$ ls > listare

ubuntu@ubuntu:~$ cat listare
fisier1
```

Pentru a adăuga rezultatul comenzii date la sfârșitul fișierului se folosește operatorul `>>`:

```
ubuntu@ubuntu:~$ date >> listare

ubuntu@ubuntu:~$ cat listare
fisier1
Sat Aug 22 16:07:50 EEST 2009
```

În listingul de mai jos se încearcă copierea unui fișier fără specificarea destinației. Această comandă generează o eroare care este scrisă în fișierul `erori`:

```
ubuntu@ubuntu:~$ cp fisier1 2> erori

ubuntu@ubuntu:~$ cat erori
cp: missing destination file operand after 'fisier1'
Try 'cp --help' for more information.
```

Pentru a adăuga noi mesaje de eroare la sfârșitul fișierului se folosește operatorul `2>>`:

```
ubuntu@ubuntu:~$ cp fisier4 2>> erori

ubuntu@ubuntu:~$ cat erori
cp: missing destination file operand after 'fisier1'
Try 'cp --help' for more information.

cp: missing destination file operand after 'fisier4'
Try 'cp --help' for more information.
```

Comanda de mai jos scrie și rezultatele și erorile generate de comanda ls în fișierul specificat:

```
ubuntu@ubuntu:~$ ls fisier1 fisier4 2>&1 > rezultate
```

sau mai simplu:

```
ubuntu@ubuntu:~$ ls fisier1 fisier4 >& rezultate

ubuntu@ubuntu:~$ cat rezultate
ls: cannot access fisier4: No such file or directory
fisier1
```

Tabelul 4.9 prezintă câteva exemple de redirectări folosind fișiere speciale:

4.5 Drepturi de acces

O primă măsură de protecție a datelor o reprezintă drepturile de acces la fișiere. Atât timp cât un utilizator nu are drepturi de administrator pe un anumit computer, acel utilizator se supune drepturilor de acces la fișiere.

Tabelul 4.10 prezintă drepturi ce pot fi configurate pentru un fișier, fără a particulariza la un anumit tip de sistem de fișiere.

Tabelul 4.9: Comenzi de redirectare care folosesc fișiere speciale

Comandă	Efect
`./program 2>/dev/null`	mesajele de la ieșirea de erori nu sunt afișate
`./program 2>&1 >/dev/null`	niciun mesaj nu este afișat
`> fout`	creează un fișier gol cu numele `f_out`
`cat /dev/null > f_out`	creează un fișier cu același conținut ca `/dev/null`, adică un fișier gol

Tabelul 4.10: Drepturi de acces

Drept	Descriere
citire	dreptul de a deschide și citi conținutul unui fișier
scriere	dreptul de a scrie într-un fișier
execuție	dreptul de a executa un fișier (aplicație) sau, pentru directoare, dreptul de a intra într-un director
modificare	dreptul de a modifica datele dintr-un fișier existent
ștergere	dreptul de a sterge un fișier

4.5.1 Utilizatori și grupuri de utilizatori vs. liste de acces

Există două metode mai întâlnite pentru definirea drepturilor de acces la fișiere:

- drepturi de acces la nivel de utilizator/grup.
- liste de acces – *Access Control Lists* (ACL).

Prima metoda (drepturi de acces la nivel de utilizator/grup) constă în definirea unor drepturi pentru următoarele entități:

- posesorul unui fișier (*user*);
- grupul care deține fișierul (*group*);
- toți ceilalți utilizatori (*others*).

Un utilizator se poate afla în mai multe grupuri (vezi secțiunea 3.1). Această metodă este cea mai folosită cale pentru definirea drepturilor de acces. Metodă oferă un nivel de protecție suficient pentru majoritatea situațiilor, ocupând un spațiu limitat.

În cadrul sistemului bazat pe liste de acces, unui fișier i se pot asocia mai mulți utilizatori și/sau grupuri de utilizatori. Pentru fiecare dintre aceste entități pot fi configurate mai multe tipuri de drepturi. Această variantă poate fi privită ca o extindere a sistemului cu drepturi de acces prezentat anterior. Complexitatea poate face ca sistemul să fie destul de greu de întreținut.

Fiecare sistem de fișiere oferă un set de drepturi ce pot fi modificate pentru fiecare fișier în parte. Ca diferențe notabile între sistemele de fișiere se pot aminti următoarele:

- FAT32 nu oferă suport pentru drepturi de acces la fișiere; există doar posibilitatea de a marca un fișier ca read-only; orice utilizator poate să schimbe acest drept;

- majoritatea sistemelor de fișiere în mediile Unix (inclusiv Mac OS X) au suport pentru drepturi de acces bazat pe utilizator/grup și, folosind o extensie, pot fi extinse cu liste de acces;
- NTFS are un sistem foarte avansat de drepturi de acces, bazat pe liste de acces (vezi secțiunea 10.6.1); pentru fiecare entitate adăugată în lista unui anumit fișier pot fi configurate mai multe drepturi.

4.5.2 Clasificarea drepturilor de acces

Sistemele de fișiere din mediile Unix care au ca implementare standard drepturile de acces la nivel user/grup au la baza următorul set de drepturi de acces:

- **citire** (*read*) – deschidere și citire de fișiere;
- **scriere** (*write*) – creare și scriere de fișiere;
- **execuție** (*execute*) – execuție fișiere/intrare în directoare.

În aceste sisteme de fișiere exista 3 entități pentru care poate fi stabilită orice combinație a drepturilor definite mai sus:

- **utilizator** (*user*) – posesorul fișierului;
- **grup** (*group*) – grupul de care aparține fișierul;
- **alții** (*others*) – utilizatorii care nu fac parte din grupul fișierului și nici nu sunt posesori.

4.5.3 Vizualizarea drepturilor de acces

Pentru a vedea drepturile de acces pentru un anumit fișier este suficientă utilizarea comenzii **`ls`** cu parametrul `-l`. Spre exemplu, pentru a vedea drepturile de acces asupra unui fișier, transmitem ca parametru comenzii **`ls`** direct numele fișierului:

```
1 ubuntu@ubuntu:~$ ls -l fisier.txt
2 -rwxr----- 1 ubuntu users 0 2007-09-20 12:47 fisier.txt
```

Drepturile de acces sunt date de primele 10 caractere din ieșirea comenzii **`ls`**:

- primul caracter reprezintă tipul fișierului:
  - `-` = fișier obișnuit
  - `d` = director
  - `p` = pipe
  - `b` = dispozitiv bloc
  - `c` = dispozitiv caracter
  - `l` = legătură

- următoarele 3 caractere (`rwx`) reprezintă drepturile de acces pentru utilizatorul `ubuntu`, care este deținătorul fișierului; se observă că utilizatorul are toate drepturile (scriere/citire/execuție);
- următoarele 3 caractere (`r--`) reprezintă drepturile de acces pentru utilizatorii care fac parte din grupul `users`; se observă că este prezent dreptul de citire dar lipsesc drepturile de scriere și execuție;
- următoarele 3 caractere (`---`) reprezintă drepturile de acces pentru utilizatorii care nu sunt `ubuntu` și nici nu fac parte din grupul `users`; acest utilizatori nu au niciun drept.

Există două moduri de reprezentare a drepturilor:

- în formă **numerică**: pentru fiecare entitate exista o cifră în baza 8 care descrie drepturile, câte un bit pentru fiecare drept;
- în formă **literală**: drepturile sunt referite direct prin inițiala lor, pentru fiecare tip de entitate.

Astfel, pentru exemplul de mai sus, avem drepturile:

- `rwxr-----` în forma listing;
- `111100000` în forma binară;
- `740` în forma octală;
- `u=rwx,g=r` în forma literală.

Mai multe informații despre securitatea fișierelor și modul de schimbare a drepturilor de acces se găsesc în secțiunea 10.2.4.

4.6 Căutarea fișierelor

4.6.1 Comanda find

Comanda **`find`** folosește abordarea brute-force pentru găsirea fișierelor căutand în arborele de directoare. Comanda pune la dispoziție un set extins de criterii de căutare, cum ar fi numele fișierului, utilizator, grup, tip, permisiuni, dimensiune, dată, și altele. Exemple de criterii de căutare sunt prezentate în continuare specificând opțiunea corespunzătoare a comenzii **`find`**:

- `-name` – căutare după numele fișierelor
- `-perm` – căutare după permisiunile fișierelor
- `-size` – căutare după dimensiunea fișierelor

Comanda următoare caută toate fișierele care au numele `stat`, în mod recursiv în directorul `/usr`:

```
1 ubuntu@ubuntu:~$ find /usr -name stat
2 /usr/src/linux-headers-2.6.24-19-generic/include/config/cpu/freq/stat
3 /usr/bin/stat
```

Următoarea comandă caută toate fișierele care au permisiunile `644`:

```
ubuntu@ubuntu:~$ find /usr -perm 644
/usr/local/include/glib-2.0/glib.h
/usr/local/include/glib-2.0/gobject/gsourceclosure.h
/usr/local/include/glib-2.0/gobject/gtypeplugin.h
[...]
```

Pentru a căuta fișierele cu dimensiunea mai mare de 500 KB se poate folosi una dintre următoarele comenzi:

```
ubuntu@ubuntu:~$ find /usr -size +500000

ubuntu@ubuntu:~$ find /usr -size +500k
/usr/local/lib/libgio-2.0.so.0.1800.4
/usr/local/lib/libgobject-2.0.so.0.1800.4
/usr/local/lib/libglib-2.0.so.0.1800.4
[...]
```

4.6.2 Comanda locate

Comanda **`locate`** foloseste o baza de date locală în care sunt indexate toate fișierele. Comanda **`locate`** este mai rapida decât comanda **`find`**, dar pune la dispoziție un singur criteriu de căutare: numele fișierului. Un alt dezavantaj este faptul că baza de date trebuie reactualizată periodic pentru a conține informații despre fișierele noi create în sistem. Actualizarea se realizează cu ajutorul comenzii **`updatedb`**.

Comanda întoarce o listă cu toate fișierele ale cărui nume conține șirul de caractere precizat ca argument. În exemplul următor se caută toate fișierele care conțin `pwd`:

```
ubuntu@ubuntu:~$ locate pwd
/bin/pwd
/etc/.pwd.lock
/sbin/unix_chkpwd
/usr/bin/pwdx
/usr/include/pwd.h
[...]
```

Unele distribuții Linux folosesc comanda **`slocate`** în locul comenzii locate. Avantajul comenzii **`slocate`** este acela că nu permite afișarea fișierelor din directoarele în care utilizatorul nu are drepturi de acces.

4.6.3 Comanda whereis

Această comandă poate fi folosită pentru a căuta într-un set restrâns de locații din sistem, de exemplu directoarele cu fișiere binare, directoarele cu biblioteci sau directoarele cu pagini de manual. Comanda **`whereis`** nu poate fi folosită pentru a căuta în directoarele utilizatorului. Comanda va căuta toate fișierele care încep cu șirul de caractere precizat ca argument. De exemplu, pentru a localiza comanda **`ls`** folosim următoarea comandă:

```
ubuntu@ubuntu:~$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.gz
```

Comanda va afișa calea către executabil dar și calea către pagina de manual a comenzii **ls**.

4.6.4 Comanda which

Folosind comanda `which` se poate obține calea către executabilele care pot fi rulate din linia de comandă. De exemplu, pentru a afla calea către comanda **chmod** se va folosi comanda următoare:

```
1 ubuntu@ubuntu:~$ which chmod
2 /bin/chmod
```

În mod implicit, comanda **which** returnează doar prima potrivire găsita, iar pentru a afișa lista completa a potrivirilor se va folosi opțiunea `-a`.

4.6.5 Comanda type

Această comandă poate fi folosită pentru a determina modul de interpretare a unei comenzi, de exemplu comandă integrată în shell, comandă externă sau alias.

Un exemplu de comandă integrată in shell este:

```
1 ubuntu@ubuntu:~$ type cd
2 cd is a shell builtin
```

Pentru o comandă externă rezultatul comenzii type este calea către executabil:

```
1 ubuntu@ubuntu:~$ type cat
2 cat is /bin/cat
```

În cazul unui alias este afișată comanda echivalentă:

```
1 ubuntu@ubuntu:~$ type ls
2 ls is aliased to 'ls --color'
```

4.7 Tipuri de sisteme de fișiere

În timp au apărut mai multe tipuri de sisteme de fișiere. În mediul open-source s-au dezvoltat și încă se dezvoltă cele mai multe dintre ele. Câteva dintre cele mai importante sisteme de fișiere utilizate în prezent sunt menționate în tabelul 4.11, împreună cu sistemele de operare în care se întâlnesc.

Prin *nativ* se înțelege că suportul este oferit prin drivere ce însoțesc sistemul de operare (în cazul Linux însoțesc kernel-ul).

Sistemele de fișiere se pot clasifica și după locul unde se găsesc datele. Tabelul 4.12 prezintă succint această clasificare.

Tabelul 4.11: Sisteme de operare și sisteme de fișiere suportate

Sistem de fișiere	Windows	Linux	Mac OS X
FAT32	nativ	nativ	nativ
NTFS	nativ (după WinNT)	prin ntfs-3g	prin ntfs-3g
ext2/ext3	driver third-party	nativ	–
ReiserFS	aplicație third-party	nativ	–
HFS+	–	nativ	nativ
ISO9660	nativ	nativ	nativ
UDF	nativ	nativ	nativ

Tabelul 4.12: Clasificarea sistemeleor de fișiere după suportul datelor

Tip	Exemplu	Descriere
sisteme de fișiere cu suport fizic	FAT32, NTFS, Ext3, ReiserFS, HFS+	se regăsesc de obicei pe un mediu de stocare
sisteme de fișiere virtuale	procfs, devfs, SSHFS	conțin fișiere/date generate de SO (informații despre sistem) sau de o altă componentă software (alte surse)
sisteme de fișiere pentru rețea	NFS, SMB	utilizate pentru accesul la fișiere aflate în rețea

4.7.1 Integritatea datelor

Un sistem de fișiere trebuie să asigure integritatea datelor inclusiv și în cazul unei căderi (failure). De cele mai multe ori, această funcționalitate este implementată prin jurnalizare.

> **Jurnalizarea** este activitatea prin care se realizează un jurnal cu toate modificările efectuate asupra unui sistem de fișiere. Fiecare operație care se execută asupra sistemului de fișiere este întâi scrisă în jurnal. Operația poate presupune modificări în mai multe locuri din sistemul de fișiere.

Driver-ul sistemului de fișiere execută la un anumit interval de timp operațiile din jurnal pentru a actualiza starea discului. O operație se șterge din jurnal doar după ce toate modificările asociate ei au fost executate. În acest fel, în cazul apariției unei căderi (întrerupere de curent etc.), se pot executa operațiile din jurnal care au rămas neexecutate (operațiile începute și care au fost întrerupte sunt încă prezente în jurnal și vor fi reexecutate în întregime).

4.7.2 Alegerea unui sistem de fișiere

În momentul în care trebuie ales un sistem de fișiere, cele mai utilizate criterii de selecție sunt:

- **disponibilitatea** – dacă se poate folosi sistemul respectiv de fișiere în sistemul de operare ales (sau în mai multe sisteme de operare);

- **gradul de siguranță** – majoritatea sistemelor de fișiere existente în prezent au jurnalizare;
- **restricții speciale** date de modul de organizare a datelor în sistemul de fișiere; de exemplu, la FAT32 se folosesc doar 32 biți pentru a stoca dimensiunea unui fișier, deci dimensiunea maximă a unui fișier este 4GB (mai puțin decât o imagine de DVD, fapt ce îl face nepractic pentru o parte din aplicațiile multimedia);
- **optimizări de performanță**, realizate spre exemplu în funcție de tipul suportului de stocare, în funcție de dimensiunea fișierelor etc.

Tabelul 4.13 realizează o analiză sumară a caracteristicilor sistemelor de fișiere.

Tabelul 4.13: Caracteristici ale sistemelor de fișiere mai cunoscute

Tip sistem de fișiere	**Sisteme de operare**	**Dimensiune maximă fișier**	**Jurnalizare**	**Observații**
FAT32	Windows / Linux / Mac OS X	4 GB	nu	cel mai folosit sistem de fișiere – întâlnit în mod special pe USB stick-uri, fără drepturi de acces; Windows-ul limitează la creare dimensiunea unei partiții la 32GB, dar poate citi partiții mai mari realizate și formatate cu aplicații third-party folosit pentru a asigura compatibilitatea cu dispozitive sau sisteme mai vechi
NTFS	Windows / Linux / Mac OS X	16TiB	da	singurul sistem de fișiere pentru Windows recomandat de Microsoft; singurul sistem de fișiere pentru Windows cu securitate
ext2/ext3	Linux / Windows[1]	16GiB – 64TiB	ext2-nu; ext3-da	sistemul de fișiere considerat cel mai stabil datorită unei istorii de dezvoltare foarte lungi; ext3 e compatibil cu versiunea anterioară ext2, aduce jurnalizare față de ext2
ReiserFS	Linux	8 TiB	da	cunoscut pentru o viteză mai mare decât ext3-ul la lucrul cu multe fișiere de dimensiune mică
HFS+	Mac OS X / Linux	16 EiB	da[2]	–
ISO9660	Windows / Linux / Mac OS X	în funcție de implementare	nu	sistem de fișiere utilizat în principal pe CD-uri, organizare internă concepută pentru ca datele să fie citibile ușor
UDF	Windows / Linux / Mac OS X	16 EiB	da	sistem de fișiere utilizat în principal pe mediile optice, cu suport atât pentru scriere cât și pentru citire

4.7.3 Adresarea într-un sistem de fișiere

Din punct de vedere fizic, fișierele ocupă pe dispozitivul de stocare un spațiu bine definit. Utilizatorul nu lucrează direct cu dispozitivul de stocare și nu poate adresa un fișier după poziția fizică a acestuia pe dispozitivul de stocare. S-a introdus astfel adresarea fișierelor folosind nume. În funcție de tipul sistemului de fișiere, există o serie de restricții asupra acestor nume.

Majoritatea fișierelor au și ceea ce se numește "extensie". Aceasta extensie se regăsește după numele fișierelor și este utilizată pentru identificarea tipului acestora (exemplu format `nume.ext`. În majoritatea cazurilor această extensie are 3 caractere. Sistemele de operare de tip Windows utilizează extensia pentru a asocia un tip de fișier cu o anumită aplicație care poate deschide acel tip de fișier. În mediile Unix, extensia de obicei lipsește. Identificarea tipului fișierelor se realizează în general pe baza conținutului fișierelor.

În Unix, comanda utilizată pentru a descoperi tipul fișierelor este **`file`**:

```
1 ubuntu@ubuntu:~$ file /boot/kernel26.img
2 /boot/kernel26.img: gzip compressed data, from Unix, last modified: Sat
  Mar 3 20:28:37 2007, max compression
```

Numele fișierelor poate fi scris cu un anumit set de caractere. În prezent, majoritatea sistemelor de fișiere folosesc standardul Unicode (de obicei UTF-8 sau UTF-16) pentru codificarea caracterelor. În tabelul 4.14 sunt prezentate câteva sisteme de fișiere și restricțiile de nume asociate lor.

Dupa cum s-a menționat și la începutul acestui capitol, directoarele se utilizează pentru o mai bună organizare a fișierelor. Într-un sistem de fișiere modern, fișierele și directoarele se regăsesc într-o structură ierarhică – sub forma unui arbore. Acest arbore are un director părinte din care se poate ajunge în orice loc în sistemul de fișiere.

Calea către un fișier este formată dintr-o înșiruire de directoare, separate de caractere speciale. În mediile Unix separatorul este caracterul / (*slash*). În mediile Windows, separatorul este caracterul \ (*backslash*).

4.8 Lucrul cu sistemele de fișiere

Această secțiune tratează acțiunile cele mai cunoscute la nivelul unui sistem de fișiere. Acțiunile cele mai frecvente sunt:

- crearea unui sistem de fișiere;
- montarea unui sistem de fișiere;
- repararea unui sistem de fișiere.

Printre acțiunile rare dar care pot prezenta interes trebuie menționate:

- crearea unei imagini pentru un sistem de fișiere;
- ștergerea unui sistem de fișiere (din diferite motive).

Tabelul 4.14: Restricții asupra numelor fișierelor

Sistem de fișiere	Case sensitive	Lungime nume	Lungime cale	Observații
FAT32	nu	`8.3` sau 255 caractere cu LFN[1]	nedefinit	initial limitat la 8 caractere numele fișierului și 3 caractere extensia; FAT32 a capatat suport pentru nume lungi (LFN) pentru un fișier; FAT32 reține atât numele lung cât și versiunea scurta în format 8.3
NTFS	da[2]	254 caractere + .	32767 octeți	orice caracter Unicode poate fi folosit, mai puțin caracterele: `NULL`, `"`, `/`, `\`, `*`, `?`, `<`, `>`, `|`, `:` (utilizate în general pentru adresarea fișierelor)
ext3	da	255 octeți	nedefinit	orice byte mai puțin `NULL` poate fi folosit în numele fișierului
ReiserFS	da	4,032 octeți / 255 caractere	nedefinit	orice octet mai puțin `NULL` poate fi folosit în numele fișierului
HFS+	nu[3]	255 caractere UTF-16	nedefinit	orice caracter Unicode este permis
ISO9660	nu	`8.3` sau Joliet: 128 octeți	adâncime maximă a căii: 8 niveluri	există mai multe niveluri pentru standard, fiecare cu restricțiile lui
UDF	da	255 octeți	1023 octeți	orice octet mai putin NULL poate fi folosit în numele fișierului

Pe lângă aceste acțiuni, folosind programe speciale, se pot efectua și alte operații asupra unui sistem de fișiere, precum:

- redimensionarea unei partiții – duce implicit la o redimensionare a sistemului de fișiere (în cazul în care acest lucru nu se întâmpla, trebuie reformatată partiția);
- convertirea unui sistem de fișiere dintr-un tip în altul;
- mutarea unei partiții – de obicei nu are efecte asupra sistemului de fișiere aflat pe aceasta.

4.8.1 Crearea unui sistem de fișiere

După cum s-a menționat anterior, majoritatea sistemelor de fișiere au un suport fizic asociat unde se regăsesc efectiv datele. Suportul unui sistem de fișiere se numește

partiție. O partiție poate avea un singur sistem de fișiere. Partiționarea a fost descrisă în secțiunea 2.2.2.

Sistemul de fișiere este plasat pe o partiție prin formatare. Formatarea este procesul prin care pe partiție sunt scrise structuri de date specifice sistemului de fișiere. Aceste structuri sunt utilizate de driver-ul asociat sistemului de fișiere pentru a identifica și utiliza datelor scrise pe partiție.

Într-un sistem Linux, formatarea se realizează prin intermediul utilitarului **`mkfs`** cu sintaxa:

```
1 root@ubuntu:~# mkfs -t \textless{}tip_sistem_de_fisiere> \textless{}
  dispozitiv>
```

Exemplu de utilizare:

```
1 root@ubuntu:~# mkfs -t ext3 /dev/sda2
```

Comanda prezentata anterior formatează partiția a 2-a de pe primul disc utilizând sistemul de fișiere ext3. Pentru fiecare tip de sistem de fișiere există un program pentru formatare. Aceste programe au numele de forma **`mkfs.<fstype>`** și pot fi apelate direct pentru a formata o partiție. Comanda echivalentă exemplului anterior este următoarea:

```
1 root@ubuntu:~# mkfs.ext3 /dev/sda2
```

O operație suplimentară formatării este verificarea sectoarelor din cadrul partiției. Dacă un sector este defect, se va marca și nu va fi folosit pentru stocarea datelor. Următoarea comandă va determina verificarea sectoarelor și formatarea partiției `/dev/sda2`:

```
1 root@ubuntu:~# mkfs.ext3 -c /dev/sda2
```

Pentru a crea un sistem de fișiere MS-DOS pe o anumită partiție se poate folosi comanda **`mkdosfs`** care este un front-end pentru comenzile **`mkfs.vfat`** și **`mkfs.msdos`**. Cu optiunea `-F` se poate specifica tipul tabelei FAT care poate să fie pe 12, 16 sau 32 biți. Următoarea comandă va crea un sistem de fișiere FAT 32 pe partiția specificată:

```
1 root@ubuntu:~# mkdosfs -F 32 /dev/sda2
```

4.8.2 Montarea unui sistem de fișiere

Sistemele de operare oferă acces uniform la datele din toate tipurile de sisteme de fișiere prin intermediul ierarhiei de fișiere.

> Pentru ca un sistem de fișiere să fie vizibil sistemului de operare el trebuie întâi să fie montat (*mount*) într-un director, numit director de montare (*mount point*). Operația se numește **montarea unui sistem de fișiere**.

Această operație este executată de cele mai multe ori în mod automat de sistemul de operare:

- în Windows, sistemele de fișiere primesc automat litere diferite; pot fi ulterior modificate, cu anumite restricții, folosind `Control Panel > Administrative Tools > Computer Management > Storage > Disk Management`;

- în Linux, sistemele de fișiere sunt montate atât în locuri bine definite (definite de utilizator la instalare) cât și în locuri special create (de exemplu `/media/<nume>/`) pentru a găzdui sistemele de fișiere de pe medii externe, precum USB stick-uri, CD/DVD-ROM-uri etc.

În Mac OS X, sistemele de fișiere sunt montate automat în `/Volumes/<nume>`.

Montarea manuală

Pentru a monta un sistem de fișiere, într-un mediu Linux se utilizează comanda **mount**, cu sintaxa:

```
root@ubuntu:~# mount -t <tip_sistem_fisiere> [-o <optiuni>] <dispozitiv>
<director_montare>
```

În exemplul de mai jos se folosește **mount** pentru a monta partiția a 2-a de pe primul disc, cu sistemul de fișiere ext3, în directorul `/mnt/data` (directorul `/mnt/data` trebuie să existe anterior execuției comenzii).

```
root@ubuntu:~# mount -t ext3 /dev/sda2 /mnt/data
```

Montarea unui sistem de fișiere doar cu drepturi de citire (*read-only*) se poate realiza prin opțiunea `-r`:

```
root@ubuntu:~# mount -r -t ext3 /dev/sda2 /mnt/data
```

Alte exemple de utilizare a opțiunilor de montare se găsesc în secțiunea 4.10.

Montarea automată

Într-un sistem Linux, fișierul `/etc/fstab` descrie sistemele de fișiere care vor fi montate automat la pornirea sistemului de operare. Fișierul conține câte o linie pentru fiecare sistem de fișiere. Comentariile încep cu `#`.

Sintaxa unei linii este:

```
<dispozitiv> <director montare> <tip sistem fisiere> <optiuni> <dump> <
pas>
```

Câmpurile au următoarele semnificații:

- `<dispozitiv>` – fișier care indica locul (de obicei un dispozitiv) unde se găsește sistemul de fișiere;
- `<director montare>` – director existent în structura de directoare unde se va putea accesa conținutul sistemului de fișiere montat;
- `<tip sistem fișiere>` – tipul sistemului de fișiere;
- `<opțiuni>` – opțiuni pentru montarea sistemului de fișiere - diferă de la un sistem de fișiere la altul;
- `<dump>` – indică dacă să se realizeze automat backup al partiției utilizând programul **dump**; opțiunea este în prezent rar folosită, preferându-se alte metode de backup; acest parametru are în mod obișnuit are valoarea 0;

- <pas> – indică ordinea în care fsck va verifica sistemele de fișiere (0 = **fsck** nu va verifica sistemul de fișiere).

Un exemplu de linie din /etc/fstab este:

```
1 /dev/sda1               /               reiserfs defaults 0 0
```

Linia descrie montarea primei partiții de pe primul disc în /, utilizând sistemul de fișiere ReiserFS cu opțiuni implicite, fără backup și fără verificări.

După efectuarea unei modificări în /etc/fstab, comanda

```
1 root@ubuntu:~# mount -a
```

va monta toate partițiile menționate.

Fără niciun argument, **mount** afișează lista sistemelor de fișiere montate la un moment dat.

```
1 ubuntu@ubuntu:~$ mount
2 /dev/sda1 on / type reiserfs (rw)
3 none on /proc type proc (rw)
4 none on /sys type sysfs (rw)
5 usbfs on /proc/bus/usb type usbfs (rw)
6 none on /dev/pts type devpts (rw)
7 none on /dev/shm type tmpfs (rw)
8 tmpfs on /tmp type tmpfs (rw)
```

Se observă că în listă nu apar doar sistemele de fișiere montate manual sau cele montate automat (descrise în /etc/fstab). Există și sisteme de fișiere care sunt create și montate automat de sistem în ierarhia de fișiere pentru a oferi informații despre starea sistemului (precum /proc – vezi secțiunea 5.1.3).

Lista sistemelor de fișiere montate la un moment dat se poate regăsi de asemenea și în fișierul /etc/mtab.

Operația inversă montării unui sistem de fișiere se numește **demontare**. Pentru a realiza demontarea se utilizează comanda **umount**, cu două sintaxe posibile:

```
1 root@ubuntu:~# umount <dispozitiv>
2
3 root@ubuntu:~# umount <director_de_montare>
```

Folosind **mount** și **grep** se află tipul partiției /dev/sda2:

```
1 root@ubuntu:~# mount | grep /dev/sda2
2 /dev/sda2 on /home type reiserfs (rw)
```

Pentru a demonta acest sistem de fișiere se poate utiliza oricare din următoarele două comenzi:

```
1 root@ubuntu:~# umount /dev/sda2
2
3 root@ubuntu:~# umount /home
```

4.8.3 Repararea unui sistem de fișiere

După cum s-a menționat și la începutul acestui capitol, sistemele de fișiere moderne au suport pentru jurnalizare. Acest suport este util în mod deosebit atunci când se produce o cădere a sistemului, precum atunci când se întrerupe alimentarea cu energie electrică.

La repornire, sistemul de operare efectuează o verificare a integrității sistemelor de fișiere. La această verificare, se execută comanda **`fsck`** pentru fiecare sistem de fișiere. Aceasta comanda este particularizată pentru fiecare sistem de fișiere cu numele **`fsck.<tip_sistem_de_fisiere>`**, precum comanda **`mkfs`**.

Pentru a verifica un sistem de fișiere comanda **`fsck`** se apelează cu următoarea sintaxă:

```
1 root@ubuntu:~# fsck -t <tip_sistem_fisiere> <dispozitiv>
```

Dacă se dorește verificarea celei de-a doua partiții de pe primul disc, cunoscându-se că ar trebui să se găsească un sistem de fișiere ext3 pe aceasta, se utilizează comanda:

```
1 root@ubuntu:~# fsck -t ext3 /dev/sda2
```

În funcție de opțiunile care se folosesc la execuția comenzii, **`fsck`** poate repara erorile pe care le întâlnește.

4.8.4 Crearea unei imagini pentru un sistem de fișiere

Imaginea unui sistem de fișiere reprezintă copia fidelă într-un fișier a fiecărui octet care descrie sistemul de fișiere (inclusiv datele din acesta). Această imagine poate fi ulterior copiată pe o altă partiție sau arhivată pentru o eventuală recuperare a datelor dacă partiția sursă este distrusă accidental. O utilizare foarte frecventă a imaginilor se întâlnește în copierea conținutul CD/DVD-urilor pe alte medii de stocare.

Pentru a realiza o imagine a unui sistem de fișiere se utilizează comanda **`dd`** (vezi secțiunea 7.6.5. În exemplul de mai jos se copiază conținutul primei partiții (sistemul de fișiere) de pe discul al 2-lea în fișierul `/mnt/data/partitie1.img`.

```
1 root@ubuntu:~# dd if=/dev/sdb1 of=/mnt/data/partitie1.img
```

Această comandă este independentă de sistemul de fișiere căruia i se face imaginea. Lucrează direct la nivelul octeților care alcătuiesc sistemul de fișiere și păstrează întreaga structură internă a acestuia.

În cazul utilizării comenzii pe sisteme de fișiere foarte mari, se va observa că timpul de execuție este destul de mare. Motivul pentru care se întâmplă acest lucru este faptul că, în mod implicit, comanda **`dd`** lucrează cu blocuri de date de 512 octeți, ceea ce înseamnă ca pentru fiecare 512 octeți comanda execută o citire și o scriere. În prezent, buffer-ele discurilor au valori dimensiuni mult mai mari și pot lucra cu blocuri mai mari de 512 octeți.

Pentru a modifica dimensiunea blocului citit/scris de dd se utilizează parametrul `bs=<dimensiune>`, unde `<dimensiune>` poate să fie `xKB`, `xMB` etc.

Spre exemplu:

```
1 root@ubuntu:~# dd bs=2MB if=/dev/sdb1 of=/mnt/data/partitie1.img
```

va executa aceeași operație dar citind blocuri de 2MB.

Există și un dezavantaj al utilizării unor blocuri de date mai mari. În anumite condiții, nu se va putea face copierea unui bloc întreg, din diferite motive precum un defect hardware al suprafeței magnetice, sau ultimul bloc nu are dimensiunea de 2MB (deoarece dimensiunea totală nu este multiplu de 2MB). De cele mai multe ori se dorește recuperarea cât mai multor date de pe disc, de preferat toate. În aceste condiții trebuie determinat daca o viteza mai mare (dimensiune mai mare a blocului) este o decizie acceptabila știind ca se pot pierde date mai multe la citire.

Pentru recuperarea datelor de pe discuri cu defecte hardware, se poate folosi programul **ddrescue**[1]. **ddrescue** este o aplicație asemănătoare **dd** dar care este specializată în recuperarea cât mai multor date de pe discuri.

4.8.5 "Ştergerea" unui sistem de fișiere

Exista două tipuri de ștergere a unui sistem de fișiere:

- ștergerea logică
- ștergerea fizică

Ştergerea logică se obține prin ștergerea partiției corespunzătoare sistemului de fișiere din tabela de partiții (descris în secțiunea 2.2.2). Din moment ce nu mai există partiție care să încapsuleze sistemul de fișiere, acesta se poate considera șters. Practic, datele lui încă mai rezidă pe disc, până la o suprascriere cu alte date.

Faptul ca datele nu sunt șterse fizic de pe disc a pus și încă pune probleme oamenilor care doresc ca datele odată ce sunt șterse să rămână șterse pentru eternitate. Din acest motiv, pentru a elimina orice posibilitate de a recupera un sistem de fișiere sau datele dintr-un sistem de fișiere, se utilizează ștergerea fizică. Ştergerea fizică este realizată de fapt prin suprascrierea blocurilor de pe suportul fizic (unde era înainte sistemul de fișiere) cu date aleatoare sau cu zerouri.

Pentru suprascriere se utilizează cel mai rapid tot programul **dd** prezentat anterior dar, în loc ca partiția să fie folosită ca sursă, ea este folosită ca destinație pentru un set de date aleatoare (sau zerouri).

Pentru a suprascrie cu zerouri o partiție, se poate folosi comanda:

```
1 root@ubuntu:~# dd if=/dev/zero of=/dev/sda1
```

Pentru a suprascrie o partiție cu date aleatoare, se folosește comanda:

```
1 root@ubuntu:~# dd if=/dev/urandom of=/dev/sda1
```

4.8.6 Monitorizarea utilizării discului

O problemă care nu poate fi neglijată este umplerea hardisk-ului. Pentru a evita acest lucru, se folosesc comenzi care să afișeze cât spațiu este consumat, de exemplu comenzile **df** și **du**.

[1] http://www.gnu.org/software/ddrescue/ddrescue.html

Comanda **df** oferă informații despre dimensiunea totală, spațiul utilizat, spațiul liber și procentul de folosire al fiecărei partiții.

```
ubuntu@ubuntu:~$ df
Filesystem   1K-blocks      Used Available Use\% Mounted on
/dev/sda1     5859784  4449900   1409884 76\% /
/dev/sda2     15361340 10174596   5186744 67\% /home
```

Atunci când procentul de folosire depașește 80% se recomandă curățarea partiției.

Se poate folosi opțiunea `-a` pentru a include și pseudo sistemele de fișiere care sunt afișate cu dimensiunea 0, de exemplu `/proc`, `/sys`:

```
ubuntu@ubuntu:~$ df -a
Filesystem           1K-blocks      Used Available Use\% Mounted on
/dev/sda1                 5859784   4449900   1409884  76\% /
/dev/sda2                15361340  10174596   5186744  67\% /home
/proc                        0         0         0   -  /proc
/sys                         0         0         0   -  /sys
```

Comanda **du** caută în mod recursiv în directoarele specificate pentru a afișa cât spațiu consumă fiecare director. În exemplul de mai jos se afișează spațiul ocupat de directorul `/etc/network`:

```
ubuntu@ubuntu:~$ du /etc/network
8       /etc/network/if-down.d
12      /etc/network/if-post-down.d
12      /etc/network/if-pre-up.d
24      /etc/network/if-up.d
64      /etc/network
```

Pentru afișarea dimensiunii în format mai ușor de citit se poate folosi opțiunea `-h`:

```
ubuntu@ubuntu:~$ du -h /etc/network
8.0K    /etc/network/if-down.d
12K             /etc/network/if-post-down.d
12K             /etc/network/if-pre-up.d
24K             /etc/network/if-up.d
64K             /etc/network
```

Pentru a opri căutarea recursivă și a afișa doar spațiul ocupat de directorul transmis ca argument se folosește opțiunea `-s`:

```
ubuntu@ubuntu:~$ du -h -s /etc/network
64K             /etc/network
```

4.9 Tendințe în sistemele de fișiere

4.9.1 Sisteme de fișiere în userspace

Sistemele de fișiere pot fi citite/scrise de sistemele de operare doar dacă exista driver-e care să le implementeze. Implementarea unui driver pentru un sistem de fișiere nu este tocmai ușoară. Acesta este și principalul motiv pentru care exista destul de puține sisteme de fișiere care să implementeze un set de funcționalități acceptabil. Majoritatea sistemelor de fișiere prezente în sistemele de operare au mulți ani de dezvoltare.

După cum s-a prezentat anterior, există situații în care s-a putut foarte ușor abstractiza accesul la o serie de date prin intermediul fișierelor (precum informații despre sistem, în `/proc`). Practic, aproape orice informație poate fi prezentată sub forma de fișier. Singura problemaă care a împiedicat mult timp această trecere către reprezentarea de date folosind fișiere a fost dificultatea scrierii unui driver adecvat.

În ultima perioadă a luat amploare utilizarea de drivere pentru diferite sisteme de fișiere în user-space, spre deosebire de driverele clasice care funcționează în kernel-space. Driverele din userspace se bazează pe faptul că există un driver în kernel care se ocupă de translatarea apelurilor la date între kernel și sistemul de fișiere din userspace. Acest suport pentru sisteme de fișiere în userspace este prezent în kernel-ul Linux cu numele FUSE[1] (Filesystem în USErspace) iar în Mac OS X cu numele MacFUSE[2].

Tabelul 4.15 prezintă avantajele și dezavantajele sistemelor de fișiere în user-space:

Tabelul 4.15: Avantaje/dezavantaje sisteme de fișiere în user-space

Avantaje	**Dezavantaje**
suport pentru scrierea de driver-e în numeroase limbaje de programare; dezvoltare mult mai rapida datorită atât limbajelor de nivel înalt cât și datorită posibilității de testare mult mai rapidă (+ debugging); nu blochează sistemul în cazul unei proaste funcționări; compatibilitate a implementărilor între Linux și Mac OS X	mai lente decât sistemele de fișiere normale; au o flexibilitate mai mică în ceea ce privește nivelul de acces la hardware, comparativ cu un driver de kernel (de obicei nu este o limitare foarte importantă, dat fiind scopul sistemelor de fișiere de acest tip)

Printre cele mai folosite sisteme de fișiere în userspace se numără:

- NTFS-3G – sistem de fișiere care oferă suport read/write pentru NTFS;
- sshfs – sistem de fișiere care oferă acces la fișierele de pe altă mașină prin intermediul SSH ca și cum ar fi parte din sistemul de fișiere local;
- EncFS – Encrypted Filesystem – sistem de fișiere care criptează automat fișierele din directorul pe care îl folosește ca suport;
- AVFS – Anti-Virus Filesystem – sistem de fișiere care realizează scanarea pentru viruși în momentul în care se realizează un acces la fișiere; este implementat ca sistem de fișiere intermediar pentru accesul protejat la alt sistem de fișiere.

Fiecare dintre aceste sisteme de fișiere se montează într-un mod diferit, datorită modului în care au fost gândite să lucreze cu datele: unele au nevoie de un nume de utilizator și o parolă, altele au nevoie de calea către un alt sistem de fișiere etc. Mai multe informații se pot găsi pe paginile web ale fiecărui proiect.

[1] http://fuse.sourceforge.net/
[2] http://code.google.com/p/macfuse/

4.9.2 ZFS, ext4, btrfs

ZFS (*Zettabyte File System*) este un sistem de fișiere pe 128 de biți și logical volume manager care prezintă multe facilitați avansate cum ar fi clone copy-on-write, snapshot-uri, dynamic stripping, verificare continuă a integrității datelor, backup și restaurare rapidă, compresie incorporată, scalabilitate. ZFS a fost dezvoltat de Sun Microsystems ca software open-source și este sub licența CCDL.

ext4 este un sistem de fișiere jurnalizat dezvoltat ca îmbunătățire a lui ext3. ext4 aduce nou suport pentru sisteme de fișiere foarte mari (până la 1 exabyte = 1024 petabytes), extents (posibilitatea de a mapa blocuri mari contigue pentru un singur fișier), compatibilitate înapoi/înainte. Compatibilitatea înapoi este dată de faptul că un sistem ext3 poate fi montat ca un sistem ext4. Compatibilitatea înainte este posibilă deoarece un sistem ext4 poate fi montat ca un sistem ext3. ext4 a fost inclus în kernel-ul Linux începând cu versiunea 2.6.19 dar la momentul scrierii acestei cărți el este marcat ca fiind în dezvoltare.

btrfs (*Better File System*) este un sistem de fișiere de tipul copy-on-write dezvoltat pentru Linux. La fel ca și ext4, btrfs contine suport pentru extents. O caracteristică principală a acestui sistem de fișiere este posibilitatea efectuării unor operații critice în timpul funcționării, de exemplu defragmentarea, modificarea dimensiunii partiției și verificarea consistenței. btrfs prezintă multe alte caracteristici printre care efectuarea de snapshot-uri și de backup-uri incrementale. Inițial dezvoltat de către Oracle, btrfs este licențiat GPL.

4.10 Studii de caz

4.10.1 Comenzi pentru lucrul cu fișiere în Windows

Tabelul 4.16 prezintă echivalența comenzilor de bază Windows și Linux pentru prelucrarea fișierelor.

4.10.2 Utilizarea în siguranță a comenzilor pe fișiere

Exista o serie de opțiuni pentru comenzile **`cp`**, **`mv`** și **`rm`** care fac utilizarea acestora să fie mult mai sigură din punct de vedere al pierderii datelor în mod accidental.

Fiecare comandă are printre opțiuni `-i`, o opțiune care determină comanda să ceară o confirmare pentru fiecare fișier șters/suprascris. Pentru a anula efectul acestui parametru poate fi folosit parametrul `-f`.

Fiecare comandă recunoaște opțiunea `-v`, opțiune care determină afișarea fișierelor procesate la momentul respectiv. Această opțiune poate reprezenta o metodă bună pentru a vizualiza ce anume se întâmplă în spatele comenzii, mai ales dacă se execută comanda pentru un set mare de fișiere.

Aceste două opțiuni (`-iv`) pot fi forțate odată cu toate execuțiile realizate de utilizator pentru aceste comenzi dacă se realizează alias-uri. Alias-urile sunt descrise de obicei

Tabelul 4.16: Comenzi pentru lucrul cu fișiere în Windows

Comanda Linux	Comanda Windows	Descriere
`comanda --help`	`comanda /?`	afișează informații despre comandă
`cd`	`cd`	schimbă directorul curent
`pwd`	`chdir`	afișează directorul curent
`clear`	`cls`	șterge ecranul consolei curente
`cp`	`copy`	copiază un fișier
`rm`	`del`	șterge un fișier
`ls`	`dir`	afișează conținutul directorului curent
`vim`	`edit`	editează un fișier text
`exit`	`exit`	închide shell-ul curent
`diff`	`fc`	compară două fișiere și afișează diferențele între ele
`find`	`find`	caută fișiere
`mkfs (mke2fs)`	`format`	formatează un disc
`free`	`mem`	afișează informații despre memoria liberă și cea ocupată
`mkdir`	`mkdir`	creează un nou director
`mv`	`move`	mută un fișier
`mv`	`ren`	redenumește un fișier
`date`	`time`	afișează ora sistemului

într-un fișier care este încărcat la pornirea shell-ului (precum `~/.bashrc`) (vezi secțiunea 12.10):

```
1 alias cp="cp -iv"
2 alias mv="mv -iv"
3 alias rm="rm -iv"
```

Astfel, la execuția oricărei dintre aceste comenzi, automat se vor lua în considerare și aceste opțiuni.

4.10.3 Montarea unui sistem de fișiere FAT32 astfel încât toți utilizatorii să aibă drept de scriere pe el

Pentru a realiza acest lucru se utilizează opțiunile de montare a sistemelor de fișierelor. Pentru a forța la montare drepturile fișierelor se utilizează măști (*mask*). Masca este o înșiruire de biți care dacă sunt 1 semnifică un drept care lipsește.

Exista mai multe tipuri de măști:

- `umask` – mască aplicată tuturor fișierelor noi (vezi secțiunea 10.2.4);
- `fmask` – mască aplicată tuturor fișierelor;
- `dmask` – mască aplicată tuturor directoarelor;

Folosind aceste măști se definesc opțiuni pentru montarea unei partiții FAT32 în felul următor:

```
1 root@ubuntu:~# mount -t vfat -o dmask=000,fmask=111 /dev/sda2 /mnt/media
```

Parametrul `-o dmask=000,fmask=111` specifică faptul că pentru toți utilizatorii fișierele normale trebuie să apară cu drepturi read/write iar directoarele cu toate drepturile. Trebuie menționat că pe o partiție FAT32 nu ar trebui să se găsească aplicații/script-uri ce pot fi executate din Linux. Pentru a preveni această situație, tuturor fișierelor le este îndepărtat dreptul de execuție, prin masca `111`.

Comanda de mai sus se poate transforma foarte ușor într-o linie în `/etc/fstab`:

```
1 /dev/sda2  /mnt/media   vfat  defaults,dmask=000,fmask=111   0   0
```

S-a păstrat opțiunea `defaults` pentru că aceasta activează alte opțiuni utile la montarea automată.

4.10.4 Montarea unui sistem de fișiere FAT32 astfel încât utilizatorii dintr-un anumit grup să aibă drept de scriere pe acesta

Exista situații când se dorește ca doar un grup de utilizatori să aibă acces la o partiție. În aceste condiții, posesorul fișierelor trebuie modificat în `root` iar grupul în grupul dorit. Grupul trebuie să aibă toate drepturile. În primul rând trebuie determinat identificatorul grupului (`gid`, *group id*). care se dorește a avea drepturi asupra fișierelor. Cunoscându-se numele grupului, comanda următoare va afișa identificatorul grupului.

```
1 ubuntu@ubuntu:~$ cat /etc/group | grep numegrupnumegrup:x:100:
2
```

Comanda filtrează liniile din `/etc/group` și afișează pe ecran doar liniile care încep cu numele grupului dorit. Numărul ce se găsește între caracterele `:` este identificatorul grupului.

Comanda care montează un sistem de fișiere FAT32 cu restricțiile menționate mai sus este următoarea:

```
1 root@ubuntu:~# mount -t vfat -o dmask=007,fmask=117,gid=100,uid=0 /dev/
  sda2 /mnt/media
```

Se observă că, spre deosebire de cazul anterior, au mai fost introduse două opțiuni:

- `gid=100` – identificatorul grupului care va avea drepturi asupra fișierelor;
- `uid=0` – identificatorul utilizatorului; utilizatorul root are `uid=0`.

Măștile au fost schimbate pentru a nu permite niciun fel de drept altor utilizatori în afara grupului și utilizatorului `root` (au `7` pe ultima poziție).

4.10.5 Sistem de fișiere într-un fișier

După cum s-a prezentat și în subcapitolele anterioare, în Linux se poate accesa (aproape) tot printr-un fișier. În particular, fiecare dispozitiv are asociat un fișier. Spre

exemplu, prima partiție de pe primul HDD are asociat (de obicei) fișierul `/dev/sda1`. În mod similar cu utilizarea fișierului `/dev/sda1`, orice alt fișier poate fi folosit ca suport pentru un sistem de fișiere. Două dintre motivele cele mai folosite pentru a realiza acest lucru sunt:

- experimentarea unor noi sisteme de fișiere, fără a repartiționa discul;
- încapsularea datelor într-un sistem de fișiere (eventual criptat).

În tabelul 4.17 este prezentat modul în care se poate realiza un sistem de fișiere stocat într-un fișier. Sunt prezentate comenzile atât pentru sistemul de fișiere aflat într-o partiție cât și pentru sistemul de fișiere aflat într-un fișier.

Tabelul 4.17: Sisteme de fișiere peste partiții sau peste fișiere

Acțiune	**Sistem de fișiere**	
	într-o partitie	**într-un fișier**
alocare spațiu	se partiționează discul; se obține partiția `/dev/sda1`	`dd if=/dev/zero of=~/date bs=1MB count=20` se crează astfel un fișier cu 20 blocuri, fiecare 1MB, se obține echivalentul unei partiții de 20MB
formatare	`/sbin/mkfs -t vfat /dev/sda1`	`/sbin/mkfs -t vfat ~/date`
verificare	–	`file ~/date`
montare	`mount -t vfat /dev/sda1 dir/`	`mount -t vfat -o loop ~/date dir/`
demontare	`umount dir/`	`umount dir/`

Opțiunea `-o loop` specifică sistemului de operare că partiția nu se găsește pe un dispozitiv fizic ci într-un fișier.

4.10.6 ntfs-3g

nfts-3g[1] este proiectul care oferă suport read-write pentru partiții NTFS sub Linux. Pentru instalarea ntfs-3g se folosește comanda:

```
1 root@ubuntu:~# apt-get install ntfs-3g
```

Pentru a identifica partiția NTFS se poate utiliza comanda:

```
1 root@ubuntu:~# fdisk -l | grep NTFS
```

care afișează doar liniile care conțin `NTFS` din ieșirea comenzii **`fdisk -l`**,

Pentru a monta sistemul de fișiere în mod automat, în `/etc/fstab` se adauga o linie de forma:

```
1 <dispozitiv> <director montare> ntfs-3g defaults,locale=en_US.utf8 0 0
```

[1] http://www.ntfs-3g.org/

Cuvinte cheie

- sistem de fișiere
- montare
- demontare
- fsck
- fișier
- director
- jurnalizare
- cale absolută, cale relativă
- ierarhie
- copiere, mutare, ștergere, redenumire
- arhivare
- dezarhivare
- backup
- descriptor de fișier
- redirectare
- drept de acces
- ls, pwd, cd
- cp, mv, rm
- mkdir, rmdir, touch

Întrebări

1. Sistemele de fișiere din prezent oferă un grad mai mare de siguranță împotriva căderilor de tensiune prin implementarea:
   - ❑ drepturilor de acces la nivel de utilizator și grup
   - ❑ drepturilor de acces prin liste de acces
   - ❑ suportului pentru legături hard
   - ❑ suportului pentru jurnalizare
2. Care este fișierul folosit pentru montarea automată a unui sistem de fișiere în Linux?
   - ❑ `/etc/resolv.conf`
   - ❑ `/etc/fsck`
   - ❑ `/mnt/hda5`
   - ❑ `/etc/fstab`
3. Un fișier are drepturile 653. Utilizatorul ce deține fișierul rulează comanda:

```
1 chmod a-x,u+w,g-r,o+r
```

   Care sunt noile drepturi ale fișierului?
   - ❑ 606
   - ❑ 762
   - ❑ 534
   - ❑ 451

4. Care din următoarele comenzi NU poate fi aplicată asupra unui director?
   - ❑ **`file`**
   - ❑ **`touch`**
   - ❑ **`rm -r`**
   - ❑ **`cat`**
5. Care din următoarele NU este o intrare validă în sistemul de fișiere?
   - ❑ fișier executabil
   - ❑ director al cărui nume începe cu .
   - ❑ legătură simbolică către /bin
   - ❑ interfață de rețea de looback
6. Care din următoarele sisteme de fișiere este un sistem nativ Linux?
   - ❑ `HPFS`
   - ❑ `FAT32`
   - ❑ `EXT3`
   - ❑ `UFS`
7. Un utilizator configurează un fișier cu drepturile 755. Ce utilizator are dreptul de execuție pe fișier?
   - ❑ root
   - ❑ utilizatorul care deține fișierul
   - ❑ niciun utilizator
   - ❑ toți utilizatorii
8. Ce reprezintă litera `v` din comanda de mai jos?

```
1 ubuntu@ubuntu:~$ tar czvf a.tgz dir1
```

   - ❑ crearea unei arhive
   - ❑ dezarhivarea unei arhive
   - ❑ crearea unei arhive GZIP
   - ❑ afișarea de mesaje despre arhivare
9. Care comandă este folosită pentru montarea unui sistem de fișiere în Linux?
   - ❑ mount
   - ❑ umount
   - ❑ mountfs
   - ❑ fsck

10. Pentru a putea utiliza un sistem de fișiere pe o partiție dată, prima acțiune absolut necesară care trebuie executată este:
    - ❑ ștergerea sistemului de fișiere existent anterior
    - ❑ montarea sistemului de fișiere de pe partiția respectivă
    - ❑ formatarea partiției cu sistemul de fișiere dorit
    - ❑ schimbarea drepturilor de acces pentru directorul în care este montată partiția

Capitolul 5

Procese

It's a well-known fact that computing devices such as the abacus were invented thousands of years ago. But it's not well known that the first use of a common computer protocol occurred in the Old Testament. This, of course, was when Moses aborted the Egyptians' process with a control-sea

Tom Galloway

Ce se învață din acest capitol?

- Definiția unui proces; diferența între un program și un proces
- Noțiunea de multitasking
- Ierarhia de procese într-un sistem de operare
- Comenzi de vizualizare a proceselor (ps, pstree, top)
- Daemoni; rularea proceselor în background/foreground
- Noțiunea de semnal; administrarea proceselor prin intermediul semnalelor
- Comunicarea între procese; operatorul | (pipe) de comunicare între procese
- Administrarea proceselor/serviciilor în Windows

5.1 Noțiuni introductive

În lumea sistemelor de operare Unix (și nu numai) există două concepte fundamentale: **fișierul** și **procesul**. Fișierul este folosit pentru a abstractiza informația și modul de stocare și utilizare a acesteia în sistemul de operare. Pe de altă parte, procesul este folosit pentru a abstractiza executarea sarcinilor sistemului de operare.

5.1.1 Ce este un proces?

> În definiția sa cea mai simplă, **un proces** este un program aflat în execuție. Dacă un program este considerat o entitate pasivă, un proces este considerat o entitate activă.

Definiția anterioară simplifică semnificația pe care o are un proces, încadrându-se în gradul de detaliu pe care îl atinge capitolul curent.

Procesul este unitatea de bază a sistemului de operare folosită pentru a îndeplini sarcinile indicate de utilizator. Orice solicitare a utilizatorului se traduce în crearea unui proces și rezolvarea acelei solicitări. Un proces este, așadar, **o entitate de execuție**.

5.1.2 Deosebirea dintre un proces și un program

Un **program** este doar un fișier executabil care se află pe disc. Se spune că un astfel de fișier este imaginea din care se va realiza un proces. Un program este, așadar, o entitate pasivă, care nu are asociate noțiuni de utilizare a memoriei și procesorului (elemente de bază într-un sistem de calcul). Fișierele `/bin/ls`, `/usr/bin/vi` sunt exemple de programe:

```
1  razvan@anaconda:~$ ls -l /bin/ls
2  -rwxr-xr-x 1 root root 96216 2008-06-27 03:31 /bin/ls
3
4  razvan@anaconda:~$ ls -l /usr/bin/vi
5  lrwxrwxrwx 1 root root 20 2009-02-27 15:43 /usr/bin/vi -> /etc/
   alternatives/vi
6
7  razvan@anaconda:~$ ls -l /etc/alternatives/vi
8  lrwxrwxrwx 1 root root 18 2009-04-24 09:59 /etc/alternatives/vi -> /usr/
   bin/vim.basic
9
10 razvan@anaconda:~$ ls -l /usr/bin/vim.basic
11 -rwxr-xr-x 1 root root 1644100 2009-03-19 17:32 /usr/bin/vim.basic
```

Un program conține, în cadrul fișierului executabil, două componente (secțiuni) principale:

- secțiunea de cod, ce conține instrucțiunile de rulare a programului pe procesor;
- secțiunea de date, ce conține informațiile utilizate la execuția programului;

Pe de altă parte, un **proces** pornește în momentul execuției unui program. În acel moment, sistemul de operare creează o structură (o entitate, o abstracție) denumită proces. Dincolo de informațiile conținute în *imagine* (fișierul executabil de pe disc), sistemul de operare asociază procesului o *zonă de memorie* unde își va menține datele și unde se va afla codul de executat. De asemenea, procesului i se asociază *timp de rulare* a codului pe procesor și *alte informații* utile pentru interacțiunea cu alte procese sau componente ale sistemului de operare. Toate aceste elemente formează împreună **contextul procesului**. Mai multe informații se găsesc în secțiunea secțiunea 14.7.1.

Exemple de procese sunt rularea comenzii **`ls`** sau rularea editorului **`vi`**. În momentul rulării comenzii **`ls`**, sistemul de operare folosește imaginea `/bin/ls` și creează un

proces. Acest proces este răspunzător pentru utilizarea resurselor sistemului (memorie, procesor) pentru a oferi rezultatul rulării comenzii (în cazul de față afișarea conținutului directorului curent):

```
razvan@anaconda:~$ ls
Desktop   bin    code          media     packages  public_html  websites
Download  bkup   extra-school  mystuff   people    school
Maildir   books  junk          official  projects  svn-repos

razvan@anaconda:~$ ls -l /bin/ls
-rwxr-xr-x 1 root root 96216 2008-06-27 03:31 /bin/ls

razvan@anaconda:~$ file /usr/bin/file
/usr/bin/file: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.15, stripped
```

În exemplul de mai sus, a doua comandă are ca argument executabilul folosit pentru crearea procesului. La fel și a treia comandă. Pot fi considerate exemple triviale de *self-reference*[1].

Se poate observa că un proces este o instanță de execuție a unui program. Un program poate avea însă mai multe instanțe de execuție asociate (instanțe care pot rula simultan sau succesiv). Există, așadar, o asociere many to one (mai multe la unul) între procese și programe.

5.1.3 Structura unui proces

Un proces este definit, la nivelul sistemului de operare, ca o **structură de date**. Un proces are proprietăți și dispune de resurse, cele mai importante fiind:

- **identificatorul unui proces**, numit și **PID** (*process id*). Este un număr întreg care identifică unic în cadrul sistemului de operare procesul respectv;
- **imaginea procesului**, reprezentată de programul din care ia naștere procesul. Imaginea procesului conține codul și datele necesare pentru execuție;
- **starea procesului**. Un proces se poate afla în starea de rulare (în momentul în care codul asociat se execută pe procesor) sau în așteptare (un alt proces se află în rulare);
- **zonele de memorie utilizate**, ce reprezintă regiunile de memorie folosite de proces pentru a menține codul și datele;
- **fișierele deschise**. Un proces menține o tabelă cu fișierele deschise. De obicei dimensiunea tabelei este limitată astfel încât și numărul de fișierele pe care le poate deschide un proces este limitat. (În Linux această limită este stabilită implicit la 1024).

[1] http://http://en.wikipedia.org/wiki/Indirect_self-reference

Vizualizarea structurii unui proces utilizând procfs

În Linux există mai multe metode de a obține informații despre structura unui proces. Una din aceste metode este analiza **procfs**. Acesta este un sistem de fișiere virtual care oferă acces la informații din sistemul de operare. Pentru a analiza un anumit proces va trebui accesat directorul `/proc/pid`. `pid` este numărul ce identifică procesul (*process id*). În exemplul de mai jos a fost accesat directorul procesului cu pid-ul 6740:

```
1 razvan@asgard:~$ cd /proc/6740/
2
3 razvan@asgard:/proc/6740$ ls
4 auxv     cwd      exe  maps  mounts      oom_adj    root   stat   status
   wchan
5 cmdline  environ  fd   mem   mountstats  oom_score  smaps  statm  task
```

Diversele intrări din acest director oferă informații legate de proces:

- pid-ul este dat de numele directorului: **6740**;
- imaginea procesului este dată de legătura simbolică **exe**:

```
1 razvan@asgard:/proc/6740$ ls -l exe
2 lrwxrwxrwx 1 razvan razvan 0 2009-07-19 19:38 exe -> /usr/bin/vim.
  gnome
```

  După cum se vede, procesul de față este o instanță a **vim** în mediu grafic (GNOME).

- linia de comandă utilizată pentru crearea procesului:

```
1 razvan@asgard:/proc/6740$ cat cmdline
2 vi 1.txt
```

  Se observă că s-a rulat comanda **vi 1.txt**.

- directorul curent – cel în care s-a rulat comanda – este dat de legătura simbolică **cwd**:

```
1 razvan@asgard:/proc/6740$ ls -l cwd
2 lrwxrwxrwx 1 razvan razvan 0 2009-07-19 19:38 cwd -> /home/razvan/
  tmp/fd_test
```

  Rezultă că directorul curent al comenzii este `/home/razvan/tmp/fd_test`.

- zonele de memorie ale procesului sunt date de fișierul maps:

```
1 razvan@asgard:/proc/6740$ cat maps
2 08048000-081ce000 r-xp 00000000 03:03 180465     /usr/bin/vim.gnome
3 081ce000-081db000 rw-p 00186000 03:03 180465     /usr/bin/vim.gnome
4 081db000-08287000 rw-p 081db000 00:00 0          [heap]
5 [...]
6 b7fdc000-b7fde000 rw-p 00014000 03:03 178741     /lib/ld-2.3.6.so
7 bfc7a000-bfc8f000 rw-p bfc7a000 00:00 0          [stack]
```

  Nu vom insista pe aceste regiuni întrucât depășesc aria de cuprindere a capitolului de față.

- fișierele deschise de proces pot fi vizualizate prin accesarea subdirectorului `fd`:

```
1  razvan@asgard:/proc/6740$ cd fd
2
3  razvan@asgard:/proc/6740/fd$ ls
4  0  1  2  3  5  7
5
6  razvan@asgard:/proc/6740/fd$ ls -l
7  total 6
8  lrwx------ 1 razvan razvan 64 2009-07-19 19:48 0 -> /dev/pts/3
9  lrwx------ 1 razvan razvan 64 2009-07-19 19:48 1 -> /dev/pts/3
10 [...]
11 lrwx------ 1 razvan razvan 64 2009-07-19 19:48 7 -> /home/razvan/tmp
   /fd_test/.1.txt.swp
```

Nu vom insista pe descrierea tuturor intrărilor, dar se poate observa că fișierul cu numărul 7 este o legătură simbolică către `/home/razvan/tmp/fd_test/.1.txt.swp`. Acest fișier este copia de lucru a editorului vim pentru fișierul `1.txt`.

Informații detaliate despre procfs sunt descrise în secțiunea 5.1.3. Sistemul de fișiere procfs este folosit drept suport de comenzile care oferă informații despre procese. Astfel de comenzi sunt **`ps`**, **`pstree`**, **`pgrep`** sau **`top`**, despre care vom discuta în secțiunile următoare.

5.1.4 Stările unui proces. Multiprogramare și multiprocesare

Într-un sistem de operare rulează, simultan, mai multe procese. Totuși un singur proces se poate executa la un moment dat pe un procesor *single-core* (sau, cu alte cuvinte, un procesor *single-core* poate executa codul asociat unui singur proces). Pe un sistem *multi-core*[1] vor putea rula mai multe procese în același timp, numărul acestora fiind egal cu numărul de nuclee de procesare ale sistemului hardware[2].

Orice proces are asociată o stare de activitate. În mod obișnuit există mai multe stări în care se poate găsi un proces, dar noi ne vom referi doar la două:

- **rulare** (*running*): procesul rulează în acel moment pe unul din procesoarele sistemului;
- **așteptare** (*waiting*): procesul așteaptă eliberarea procesorului pentru a putea rula pe acesta.

Se observă, așadar, că este nevoie de un algoritm care să ofere fiecărui proces ocazia de a fi executat pe unul din procesoarele sistemului.

> Mecanismul de alocare a proceselor pe procesoarele sistemului poartă numele de **planificare** (*scheduling*). Subsistemul nucleului care se ocupă de alocare se numește **planificator de procese** (*scheduler*).

Rolul planificatorului este de a alege un proces care să ruleze pe un procesor și, la îndeplinirea unei condiții date, să suspende execuția procesului pentru a îl înlocui cu un

[1] http://en.wikipedia.org/wiki/Multi-core_(computing)

[2] Sistemul hardware poate avea un procesor cu mai multe nuclee sau mai multe procesoare, fiecare cu mai multe nuclee

altul. Înlocuirea unui proces cu un alt proces se numește **schimbare de context** (*context switch*).

La începutul istoriei sistemelor de operare, dacă un utilizator dorea rularea unui program, el încărca pe un suport conținutul programului și aștepta rularea acestuia (în contextul unui proces). Când procesul era planificat pe procesor, acesta rula complet până la încheierea absolută a execuției. Abordarea prezentată are dezavantajul lipsei de interactivitate: dacă un proces rulează foarte mult sau dacă trebuie să comunice des cu dispozitive periferice lente, atunci alte procese care doresc să ruleze vor trebui să aștepte un timp îndelungat.

Următorul pas în evoluția sistemelor de operare a fost introducerea noțiunii de **multiprogramare**. Înainte de a intra în detalii, trebuie amintită diferența de viteza dintre procesor/memorie și dispozitivele periferice. Astfel, dacă un proces accesează hard disk-ul, va trebui să aștepte foarte mult timp pentru ca informația să fie disponibilă (numeric, diferența dintre viteza unui procesor și viteza de acces la datele de pe un hard disk este de ordinul milioanelor).

În abordarea multiprogramată, mai multe procese coexistă în cadrul unui sistem de operare. În momentul în care un proces accesează un dispozitiv periferic lent, va trebui să aștepte furnizarea informației dorite; drept urmare, procesul respectiv este suspendat temporar, și un alt proces este planificat în locul său pe procesor (*context switch*). În acest fel se mărește interactivitatea sistemului. Dacă un proces se blochează în comunicația cu un dispozitiv periferic lent, acesta nu mai irosește timpul procesorului în așteptare, ci este înlocuit cu un alt proces.

Următorul pas a fost apariția conceptului de **multiprocesare** sau **multitasking**. Acesta a venit ca răspuns la cerințele din ce în ce mai mari de interactivitate a sistemului de operare în raport cu utilizatorul. Dezavantajul multiprogramării este că un proces care rulează mult timp pe procesor fără a executa o acțiune blocantă nu va lăsa alte procese să ruleze. Acest lucru înseamnă pierderea interactivității[1]. Soluția este asocierea, cu fiecare proces, a unui timp limită de rulare pe procesor (o **cuantă de timp**). La încheierea cuantei de timp, procesul va fi automat suspendat temporar și înlocuit cu un altul. În abordarea multitasking, schimbarea de context se va produce când procesul care rulează pe procesor execută o acțiune blocantă sau când își încheie cuanta de timp.

Toate sistemele de operare moderne sunt sisteme multiprocesare (multitasking).

5.2 Ierarhia de procese în Unix. Vizualizarea proceselor sistemului

În sistemele Unix (ca atare și în Linux), procesele sunt organizate ierarhic. Astfel, se poate spune despre un proces că este procesul **părinte** al unui alt proces; viceversa, se spune că un proces este procesul **fiu** pentru un altul. Un lucru de reținut este faptul că un proces are un singur proces părinte, dar poate avea mai multe procese fiu.

Modelul ierarhiei de procese în Unix este prezentat în figura următoare:

[1] Denumirea tehnică este *starvation*

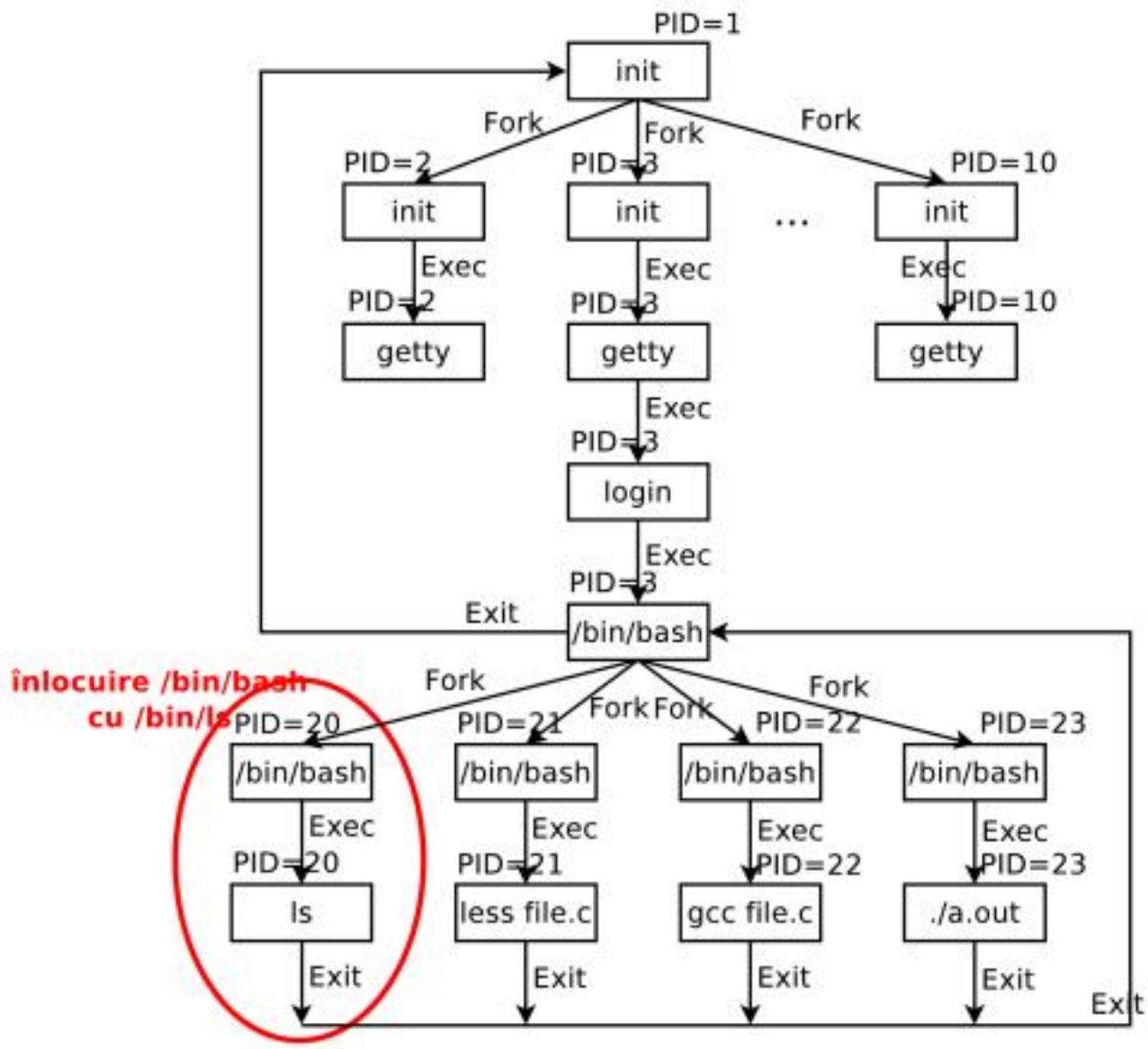


Figura 5.1: Ierarhia de procese în Unix

Ierarhia apare ca urmare a faptului că un proces este creat prin intermediul unui alt proces. La crearea ("nașterea") unui proces se realizează o copie a procesului părinte și se înlocuiește imaginea de program a părintelui cu cea dorită. Astfel, la rularea comenzii ls din shell (în imagine `/bin/bash`), se creează o copie a procesului shell, și se înlocuiește imaginea de executabil `/bin/bash` cu imaginea `/bin/ls`.

În Unix, părintele tuturor proceselor este denumit **init**. Imaginea de executabil asociată este `/sbin/init`. Acesta este primul proces creat de sistemul de operare. Este un proces special pentru că nu are părinte. Din init se creează alte procese, care la rândul lor vor crea alte procese, completându-se astfel ierarhia de procese ale sistemului.

5.2.1 Utilitarul ps

Comanda **ps** este principala comandă de vizualizare a proceselor sistemului. Informațiile prezentate de **ps** sunt obținute analizând informațiile din procfs. La o rulare simplă (fără argumente), **ps** afișează informații despre procesele pornite de utilizatorul curent pe terminalul activ:

```
1 razvan@asgard:~$ ps
2   PID TTY          TIME CMD
3  3036 pts/0    00:00:00 bash
4  3079 pts/0    00:00:00 ps
```

Rularea comenzii presupune afișarea unui cap de tabel și a unei intrări pentru fiecare proces. La o rulare simplă (fără argumente) se vor afișa identificatorul de proces (`PID`),

terminalul utilizat (`TTY`), timpul de rulare a procesului (`TIME`) și numele programului ce stă la baza acestuia (`CMD`). În exemplul de mai sus, utilizatorul a deschis un terminal cu un interpretor de comenzi (`bash`) și apoi a rulat comanda **`ps`**. Se observă că un proces **`ps`** se afișează și pe sine.

Comanda **`ps`** acceptă diferite tipuri de opțiuni care pot fi uneori conflictuale. În continuare vom insista pe opțiuni UNIX, caracterizate prin faptul că sunt precedate de semnul minus (`-`). Pentru detalii complete legate modul de utilizarea al comenzii și de efectul opțiunilor trebuie consultată pagina de manual asociată (**`man ps`**).

De exemplu, pentru afișarea proceselor pornite de utilizatorul curent, indiferent de terminalul utilizat, se folosește opțiunea `-a`.

```
1  razvan@asgard:~$ ps -a
2    PID TTY          TIME CMD
3   3074 pts/1    00:00:00 vi
4   3094 pts/0    00:00:00 ps
```

Se observă că, în această situație, se afișează procesele ce rulează pe terminalul `pts/1` și cele ce rulează pe `pts/0` fără a se mai preciza procesele tip shell de pe aceste terminale.

Comanda **`ps`** are un număr impresionant de argumente care pot controla conținutul și formatul afișării informațiilor despre procese. Vom prezenta cele două clase importante de opțiuni: opțiuni de selecție și opțiuni de formatare.

Opțiuni de selecție

Opțiunile de selecție presupun selectarea anumitor procese din sistem în funcție de diverse criterii. Sunt prezentate, în continuare, câteva opțiuni utile:

Selectarea tuturor proceselor din sistem: se realizează folosind argumentul `-e` sau `-A`:

```
1  razvan@asgard:~$ ps -e
2    PID TTY          TIME CMD
3      1 ?        00:00:00 init
4      2 ?        00:00:00 ksoftirqd/0
5      3 ?        00:00:00 events/0
6  [...]
7   3057 pts/1    00:00:00 bash
8   3074 pts/1    00:00:00 vi
9   3109 pts/2    00:00:00 bash
10  3131 pts/0    00:00:00 ps
```

Selecția proceselor după o anumită componentă: pid, comandă, terminal, utilizator; această selecție se realizează prin utilizarea unui argument specific (de exemplu `-C`, `-p` sau `-u`) urmat de lista de selecție (elementele listei sunt separate prin virgulă, fără spații între ele):

```
1  razvan@asgard:~$ ps -C bash,vi,gnome-terminal
2    PID TTY          TIME CMD
3   3034 ?        00:00:02 gnome-terminal
4   3036 pts/0    00:00:00 bash
5   3057 pts/1    00:00:00 bash
6   3074 pts/1    00:00:00 vi
```

```
 3109 pts/2    00:00:00 bash
 3180 pts/3    00:00:00 bash
```

În exemplul de mai sus au fost selectate procesele al căror executabil (`-C`) a fost `bash`, `vi` sau `gnome-terminal`.

```
razvan@asgard:~$ ps -p 1,3026,3074
  PID TTY          TIME CMD
    1 ?        00:00:00 init
 3026 ?        00:00:42 gedit
 3074 pts/1    00:00:00 vi
```

Cu comanda de mai sus s-au selectat procesele cu pid-ul (`-p`) 1, 3026 sau 3074.

```
razvan@asgard:~$ ps -u www-data
  PID TTY          TIME CMD
 2787 ?        00:00:00 apache2
 2788 ?        00:00:00 apache2
 2790 ?        00:00:00 apache2
```

În exemplul anterior s-au selectat procesele ce aparțin utilizatorului (`-u`) `www-data`.

Observați că deși s-a făcut selecția după utilizator, numele acestuia nu apare în rezultatul comenzii. Pentru aceasta vor trebui folosite opțiunile de formatare **`ps`** pentru afișarea altor câmpuri în afara celor implicite.

Opțiunile de selecție pot fi combinate. În exemplul de mai jos se selectează procesele `bash` sau `vi`, cele care au pid-ul 2968 sau 2972 și cele care aparțin utilizatorului `haldaemon`:

```
razvan@asgard:~$ ps -C bash,vi -p 2968,2972 -u haldaemon
  PID TTY          TIME CMD
 2583 ?        00:00:02 hald
 2590 ?        00:00:00 hald-addon-acpi
 2968 ?        00:00:04 metacity
 2972 ?        00:00:00 nautilus
 3036 pts/0    00:00:00 bash
 3057 pts/1    00:00:00 bash
 3074 pts/1    00:00:00 vi
 3109 pts/2    00:00:00 bash
 3180 pts/3    00:00:00 bash
```

Argumentele listei de selecție pentru opțiunile **`ps`** sunt separate prin virgulă, fără spații.

Se pot nega opțiunile de selecție cu ajutorul argumentului `-N`. Astfel, dacă se dorește selecția proceselor care nu aparțin utilizatorului privilegiat (`root`) și nici utilizatorului `razvan`, și care nu au ca executabil `apache2`, se va folosi comanda următoare:

```
razvan@asgard:~$ ps -N -u root,razvan -C apache2
  PID TTY          TIME CMD
 2283 ?        00:00:00 portmap
 2575 ?        00:00:00 dbus-daemon
 2583 ?        00:00:03 hald
 2590 ?        00:00:00 hald-addon-acpi
 2655 ?        00:00:00 exim4
 2663 ?        00:00:00 fbguard
 2664 ?        00:00:00 fbserver
 2741 ?        00:00:00 rpc.statd
 2756 ?        00:00:00 atd
```

Opțiuni de formatare

Opțiunile de formatare pentru comanda **ps** controlează coloanele care sunt afișate ca rezultat al rulării. Implicit, rularea comenzii oferă un cap de tabel cu elementele `PID`, `TTY`, `TIME` și `CMD`, urmat de o listă a proceselor. Aceste elemente pot fi schimbate prin intermediul opțiunilor de formatare.

Afișarea tuturor opțiunilor frecvente pentru **ps**:

```
1  razvan@asgard:~$ ps -F
2  UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
3  razvan    3036  3034  0  1574  3352   0 12:09 pts/0    00:00:00 bash
4  razvan    3248  3036  0   964   940   0 13:02 pts/0    00:00:00 ps -F
```

Se observă că s-a modificat capul de tabel. Sunt prezentate mai mult opțiuni utile, printre care `UID` (utilizatorul ce a creat procesul), `PPID` (pid-ul procesului părinte), `STIME` (timpul rulării procesului), `CMD` (comanda completă de rulare – cu argumente) etc.

În mod evident, opțiunile de formatare pot fi combinate cu opțiuni de selecție:

```
1  razvan@asgard:~$ ps -F -C bash,apache2,gnome-terminal,ps
2  UID        PID  PPID  C    SZ   RSS [...] TTY          TIME CMD
3  root      2786     1  0  2412  2632 [...] ?        00:00:00 /usr/sbin/
   apache2 -k start
4  www-data  2787  2786  0  2355  1948 [...] ?        00:00:00 /usr/sbin/
   apache2 -k start
5  www-data  2788  2786  0 57746  2452 [...] ?        00:00:00 /usr/sbin/
   apache2 -k start
6  www-data  2790  2786  0 57746  2456 [...] ?        00:00:00 /usr/sbin/
   apache2 -k start
7  razvan    3034     1  0  8424 15020 [...] ?        00:00:03 gnome-
   terminal
8  razvan    3036  3034  0  1574  3372 [...] pts/0    00:00:00 bash
9  razvan    3057  3034  0  1575  3348 [...] pts/1    00:00:00 bash
10 razvan    3109  3034  0  1575  3352 [...] pts/2    00:00:00 bash
11 razvan    3180  3034  0  1575  3340 [...] pts/3    00:00:00 bash
12 razvan    3257  3036  0   964   940 [...] pts/0    00:00:00 ps -F -C bash
   ,apache2,gnome-terminal,ps
```

În exemplul de mai sus, procesul cu pid-ul 2786 este procesul fiu al init (are PPID = 1). Următoarele trei procese sunt procesele fiu ale acestuia. De asemenea, procesul cu pid-ul 3034 (`gnome-terminal`) este tot un proces fiu al init, iar cele patru procese bash sunt procesele fiu ale acestuia. La fel, procesul ps (pid 3257) este procesul fiu al primului proces bash (pid 3036). O reprezentare ierarhică a celor zece procese de mai sus este următoarea:

```
1  init (1) ---> apache2 (2786)        ---> apache2 (2787)
2                                       \--> apache2 (2788)
3                                       \--> apache2 (2790)
4           \--> bash-terminal (3034) ---> bash (3036)      ---> ps (3257)
5                                       \--> bash (3057)
6                                       \--> bash (3109)
7                                       \--> bash (3180)
```

Afișarea de opțiuni specifice pentru procese

Specificarea opțiunilor de afișare pentru **ps** se realizează cu ajutorul argumentului `-o`, urmat de argumentele de formatare dorite (separate prin virgulă, fără spații).

Cele mai frecvent utilizate argumente de selecție sunt: `pid`, `ppid`, `args` (lista de argumente), `user` (utilizator), `comm` (comanda – doar numele executabilului), `cmd` (comanda cu argumente), `cputime` (timpul de procesor consumat), `group`, `pmem` (procent memorie ocupata), `state` (starea procesului – un singur caracter), `stat` (starea procesului – mai multe caractere), `tty` (terminalul de control asociat).

În continuare sunt prezentate câteva exemple.

Afișarea pid-ului (`pid`), utilizatorului (`user`), a stării procesului (`stat`), comanda cu argumente (`cmd`), timpul de procesor (`cputime`) și memoria folosită (`pmem`):

```
1  razvan@asgard:~$ ps -e -o pid,user,stat,cmd,cputime,pmem
2    PID USER     STAT CMD                              TIME %MEM
3      1 root     Ss   init [2]                      00:00:00  0.1
4      2 root     SN   [ksoftirqd/0]                 00:00:00  0.0
5      3 root     S<   [events/0]                    00:00:00  0.0
6  [...]
7   3009 razvan   S    /usr/lib/gnome-panel/notifi 00:00:00  1.4
8   3011 razvan   S    /usr/lib/gnome-applets/mixe 00:00:00  2.3
9   3016 razvan   Ss   gnome-screensaver           00:00:02  0.4
10 [...]
11  3180 razvan   Ss+  bash                        00:00:00  0.6
12  3260 razvan   R+   ps -e -o pid,user,stat,cmd, 00:00:00  0.1
```

Fără a insista pe semnificațiile stării proceselor, se observă că procesul `ps` (pid 3260) este în starea de rulare (R – running), iar celelalte sunt în starea de așteptare (S – sleeping). Sistemul de test dispune de un singur procesor și ca atare un singur proces poate rula la un moment dat.

Afișarea pid-ului, argumentelor, timpului de procesor, memoriei utilizate și timpului de pornire a comenzilor pentru procesele ce aparțin utilizatorului `www-data` și pentru procesele care au numele comenzii **bash**:

```
1  razvan@asgard:~$ ps -C bash -u www-data -o pid,args,cputime,rss,start
2    PID COMMAND                         TIME   RSS  STARTED
3   2787 /usr/sbin/apache2 -k start  00:00:00  1948 11:29:03
4   2788 /usr/sbin/apache2 -k start  00:00:00  2452 11:29:03
5   2790 /usr/sbin/apache2 -k start  00:00:00  2456 11:29:03
6   3036 bash                        00:00:00  3372 12:09:16
7   3057 bash                        00:00:00  3348 12:10:20
8   3109 bash                        00:00:00  3356 12:23:31
9   3180 bash                        00:00:00  3340 12:44:33
```

Spațiul de memorie utilizat (`rss` – resident set size), este măsurat în kiloocteți (KB).

Sortarea rezultatului rulării comenzii **ps** se realizează cu ajutorul opțiunii `--sort` urmată de lista de criterii de sortare. Semnul + (implicit) sau - în fața unui criteriu de sortare înseamnă sortare în ordine crescătoare sau descrescătoare:

```
1  razvan@asgard:~$ ps -e -o pid,cmd,cputime,pmem,rss --sort -rss
2   PID CMD                              TIME %MEM   RSS
3  3247 evince /home/razvan/school/ 00:00:37 10.7 206556
4   402 evolution                    00:01:28  7.5 146268
5  5121 /usr/bin/x-www-browser      00:01:03  5.5 107444
6  7709 pidgin                      00:00:05  2.9 56104
7  3334 /usr/bin/X :0 -audit 0 -aut 00:02:39  2.8 54960
```

```
4391 nautilus --no-default-windo 00:00:26  1.8 36324
4595 mono /usr/lib/tomboy/Tomboy 00:00:02  1.6 31252
 410 /usr/lib/evolution/evolutio 00:00:01  1.4 28844
4389 gnome-panel --sm-client-id  00:00:15  1.4 27616
[...]
```

Exemplul de mai sus realizează o sortare a proceselor din sistem după memoria ocupată. Se observă că procesele cele mai mari consumatoare de memorie sunt `evince` (un viewer de documente), `evolution` (un client de mail) și `x-www-browser` (un browser web).

Alte opțiuni

Utilitarul **`ps`** permite afișarea ierarhiei de procese în formă arborescentă. Acest lucru se realizează cu ajutorul opțiunii **`-H`**:

```
razvan@asgard:~$ ps -C init,apache2,gnome-terminal,bash -o pid,user,comm,
cputime,pmem --sort pid -H
  PID USER     COMMAND             TIME %MEM
    1 root     init            00:00:00  0.1
 2786 root       apache2       00:00:00  0.5
 2787 www-data     apache2     00:00:00  0.3
 2788 www-data     apache2     00:00:00  0.4
 2790 www-data     apache2     00:00:00  0.4
 3034 razvan     gnome-termina 00:00:05  2.9
 3036 razvan       bash        00:00:00  0.6
 3057 razvan       bash        00:00:00  0.6
 3109 razvan       bash        00:00:00  0.6
 3180 razvan       bash        00:00:00  0.6
```

În acest caz procesele fiu sunt indentate cu două caractere spațiu față de procesul părinte.

5.2.2 Utilitarul pstree

Utilitarul **`pstree`** afișează ierarhia de procese a sistemului. În cazul unei utilizări simple (fără niciun argument), comanda afișează ierarhia de procese începând de la init:

```
razvan@asgard:~$ pstree
init-+-acpid
     |-apache2-+-apache2
     |          `-2*[apache2---26*[{apache2}]]
     |-atd
     |-bonobo-activati
     |-clock-applet
     [...]
     |-gnome-terminal-+-bash
     |                |-bash---vi
     |                |-bash---man---pager
     |                |-bash---pstree
     |                |-gnome-pty-helpe
     |                `-{gnome-terminal}
[...]
```

Dacă este dorită afișarea ierarhiei de procese începând de la un anumit proces, va trebui transmis ca parametru pid-ul acelui proces:

```
razvan@asgard:~$ pstree 3034
gnome-terminal-+-bash
               |-bash---vi
               |-bash---man---pager
               |-bash---pstree
               |-gnome-pty-helpe
               `-{gnome-terminal}
```

Se poate afișa inclusiv pid-ul proceselor din ierarhie prin intermediul opțiunii `-p`:

```
razvan@asgard:~$ pstree -p 3034
gnome-terminal(3034)-+-bash(3036)
                     |-bash(3057)---vi(3074)
                     |-bash(3109)---man(3295)---pager(3302)
                     |-bash(3180)---pstree(3343)
                     |-gnome-pty-helpe(3035)
                     `-{gnome-terminal}(3037)
```

Ca și în cazul comenzii **ps**, **pstree** utilizează procfs pentru obținerea de informații despre ierarhia de procese.

5.2.3 Utilitarul pgrep

Comanda **pgrep** este echivalentul **grep** pentru lucrul cu procese și poate fi folosită pentru a afișa doar procesele din sistem care îndeplinesc o condiție. O funcționalitate echivalentă **pgrep** poate fi obținută cu ajutorul comenzii **ps** și al argumentelor specifice acesteia: `-p` pentru selecția după pid, `-C` pentru selecția după comandă, `-u` pentru selecția după utilizator etc.

În continuare sunt prezentate exemple ale rulării **pgrep** pentru selecția proceselor care au ca părinte procesul init (pid 1). Implicit se selectează doar pid-urile proceselor. Dacă se dorește și afișarea numelor proceselor, se folosește opțiunea `-l`:

```
razvan@asgard:~$ pgrep -P 1
2
3
4
5
1021
2284
[...]
3025
3027
3052

razvan@asgard:~$ pgrep -P 1 -l
2 ksoftirqd/0
3 events/0
4 khelper
5 kthread
1021 udevd
2284 portmap
[...]
3025 gnome-screensav
```

```
22 3027 gnome-terminal
23 3052 soffice
```

5.2.4 Utilitarul top

Utilitarul **top** este utilizat pentru a afișa informații în mod dinamic (în timp real) despre procesele existente în sistem. O utilizare frecventă a acesteia este monitorizarea sistemului (vezi secțiunea 10.3.1).

La o rulare fără argumente, **top** va afișa un ecran cu informații despre sistem și procesele care rulează. În mod implicit, acest ecran este actualizat la fiecare 3 secunde. Un exemplu de ecran afișat prin rularea comenzii **top** este prezentat mai jos:

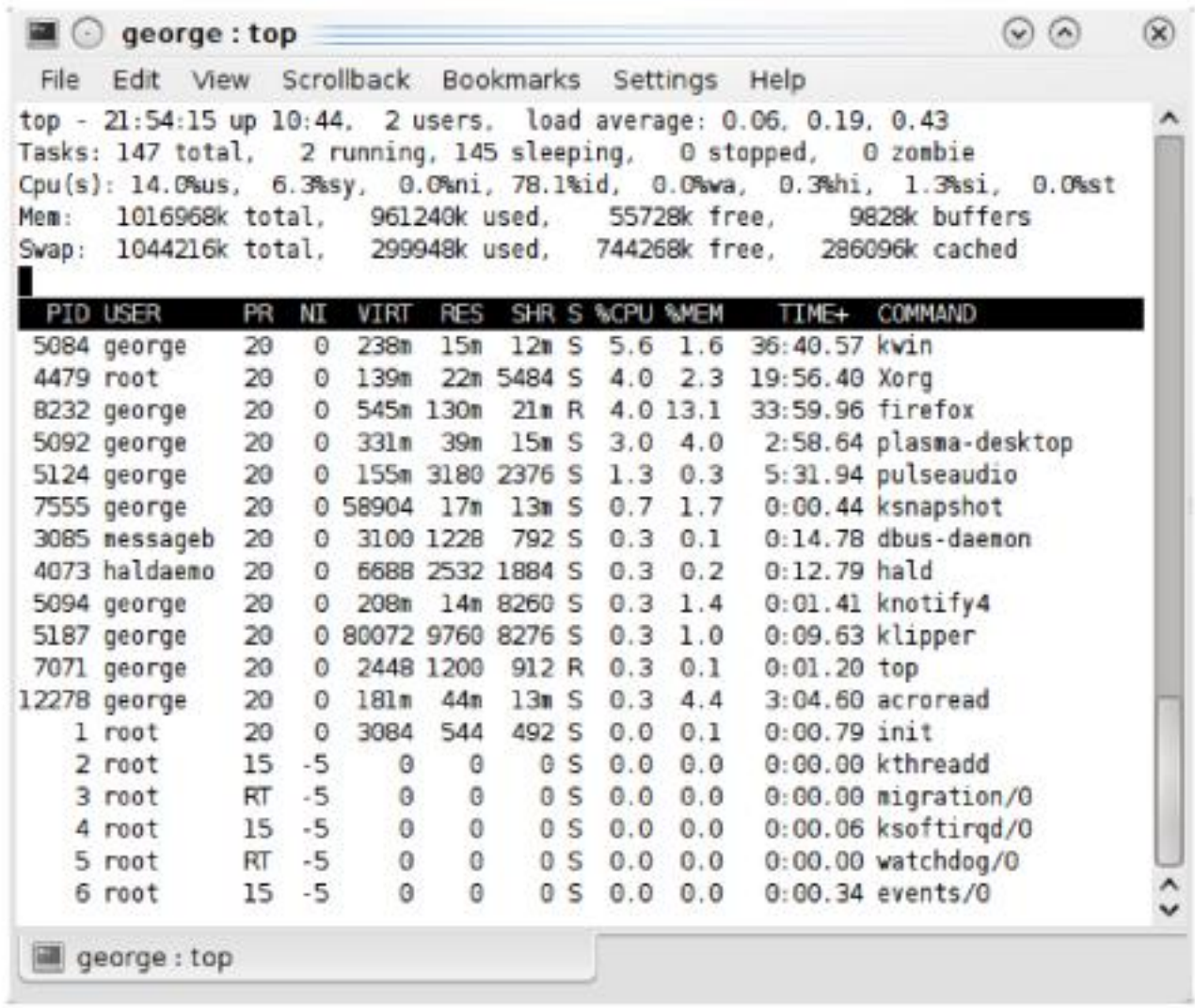


Figura 5.2: Ecran de rulare top

Ecranul afișat de top cuprinde următoarele zone:

- **zona de sumarizare** (*summary area*): aceasta conține informații generale despre sistem: încărcarea sistemului (prima linie), numărul de procese și starea lor (a doua linie), timpul de procesor utilizat (a treia linie), memoria și spațiul de swap ocupat (ultimele două linii). Gradul de încărcare a sistemului este o măsura a încărcării procesorului și poate fi determinat și cu ajutorul comenzii **uptime**.
- **zona de comandă** (*prompt line*): aici utilizatorul poate introduce comenzi specifice de interacțiune cu utilitarul **top**;
- **antetul de tabel** (*columns header*): afișează antetul de tabel pentru informațiile afișate despre procese;
- **zona de procese** (*task area*): afișează informații despre procese.

Interfața interactivă top

top are o interfața interactivă cu utilizatorul care îi permite acestuia alterarea formatului de afișare. Comanda cea mai utilă pentru lucrul interactiv cu **top** este cea de afișare a ecranului de ajutor (tasta `h`).

Majoritatea comenzilor top sunt de tipul *toggle*, adică afișarea sau oprirea afișării unei anumite componente din ecranul de afișare. În continuare vor fi prezentate câteva din comenzile top cele mai utile în funcție de zona ecranului de afișare în care au efect. Detalii suplimentare se pot afla prin consultarea paginii de ajutor (prin apăsarea tastei `h`) sau a paginii de manual (`man top`).

Comenzi pentru zona de sumarizare. Comenzile folosite pentru lucrul cu zona de sumarizare sunt comenzi de tipul *toggle*:

- tasta `l` este folosită pentru a activa/dezactiva afișarea de informații despre încărcarea sistemului (prima linie din zona de sumarizare);
- tasta `t` este folosită pentru a activa/dezactiva afișarea de informații despre numărul de procese și utilizarea procesorului (liniile 2 și 3 din zona de sumarizare);
- tasta `m` este folosită pentru a activa/dezactiva afișarea de informații despre utilizarea memoriei sistemului (ultimele două linii din zona de sumarizare).

Comenzi pentru zona de procese. Dintre comenzile care afectează zona de afișare pentru procese amintim:

- tasta `c` este folosită pentru a afișa comanda completă, nu doar numele procesului;
- tasta `f` este folosită pentru a adăuga/elimina coloane utilizate pentru afișarea de informații despre procese;
- tasta `o` este folosită pentru a schimba ordinea coloanelor din tabel; după apăsarea tastei `f` sau `o`, utilizatorului îi este prezentat un ecran de configurare, în care poate specifica noi coloane sau o nouă ordine (tot prin intermediul unor taste);
- tasta `R` (de tip *toggle*) este folosită pentru sortare ascendentă sau descendentă;
- tasta `F` sau tasta `O` sunt folosite pentru a configura coloana după care se face sortarea proceselor din zona de procese. În mod implicit sortarea se realizează după timpul de procesor, astfel încât procesele care într-un interval dat de timp au utilizat cel mai mult procesorul vor apărea primele în listă; tastele `<` și `>` pot fi folosite pentru schimbarea coloanei după care se relizează sortarea, selectând coloana din stânga sau din dreapta.

Comenzi în zona de comandă Aceste comenzi folosesc zona de comandă și afectează, de obicei, procesele sau zona de procese:

- tasta `k` permite terminarea unui proces. Apăsarea acestei taste oferă promptul `PID to kill:` unde utilizatorul va introduce pid-ul procesului a cărui execuție se dorește a fi încheiată;

- tasta `n` permite precizarea numărului de procese care vor fi afișate în zona de procese; apăsarea acestei taste conduce la apariția unui prompt unde utilizatorul precizează numărul de procese pe care dorește să le vizualizeze/monitorizeze;
- tasta `u` permite precizarea unui utilizator, fiind apoi afișate numai procesele care aparțin acestuia. Zona de comandă afișează promptul `Which user (blank for all)`: unde se cere introducerea unui nume de utilizator.

Interfața neinteractivă top

`top` poate fi utilizat și în mod neinteractiv prin folosirea argumentului `-b` în linie de comandă. Pornirea neinteractivă este folosită de obicei împreună cu argumentul `-n`, care precizează numărul de iterații de afișare. Se obține astfel un rezultat ce poate fi redirectat într-un fișier pentru analiză ulterioară:

```
1  george@asgard:~$ top -b -n 1
2  top - 23:03:58 up 11:54,  2 users,  load average: 0.11, 0.52, 0.52
3  Tasks: 148 total,   3 running, 145 sleeping,   0 stopped,   0 zombie
4  Cpu(s): 13.4%us,  6.3%sy,  2.0%ni, 77.1%id,  0.7%wa,  0.2%hi,  0.3%si,
   0.0%st
5  Mem:   1016968k total,   976164k used,    40804k free,    12684k buffers
6  Swap:  1044216k total,   395716k used,   648500k free,   371860k cached
7
8    PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
9   5124 george    20   0  155m 2656 2324 S  2.0  0.3   6:23.52 pulseaudio
10  8232 george    20   0  553m 134m  20m S  2.0 13.5  38:13.04 firefox
11 11446 george    20   0  2444 1096  824 R  2.0  0.1   0:00.01 top
12     1 root      20   0  3084  540  488 S  0.0  0.1   0:00.79 init
13     2 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kthreadd
14     3 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0
15     4 root      15  -5     0    0    0 S  0.0  0.0   0:00.07 ksoftirqd/0
16     5 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
17     6 root      15  -5     0    0    0 S  0.0  0.0   0:00.41 events/0
18     7 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 cpuset
19     8 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 khelper
20    11 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 netns
21 [...]
```

5.2.5 Timpul de execuție al unui proces. Comanda time

Timpul de execuție al unui proces se referă la durata sa de execuție în sistem, din momentul creării și până în momentul terminării acestuia.

Comanda **`time`** este utilizată pentru măsurarea timpului de rulare a unui proces și a resurselor utilizate de acesta. Rezultatul rulării implicite a comenzii **`time`** oferă 3 timpi:

- **real** – timpul efectiv de execuție, măsurat de la crearea procesului și până la terminarea acestuia;
- **user** – timpul petrecut în spațiul utilizator;
- **sys** – timpul petrecut rulând apeluri de sistem (în spațiul kernel).

time primește ca argument numele unui program a cărui durata de execuție va fi măsurată. Un exemplu de rulare este prezentat mai jos:

```
root@asgard:/home/razvan# time updatedb

real    4m19.309s
user    0m2.680s
sys     0m5.550s
```

Se observă că rularea comenzii **updatedb** a durat 4 minute și 19 secunde. Totuși timpul real de rulare a procesului se măsoară adunând timpul petrecut în spațiul utilizator (2.68 secunde) și timpul petrecut în spațiul kernel (5.55 secunde). Cu alte cuvinte, din timpul cât a durat rularea comenzii, procesul a consumat doar 8.23 secunde pentru a fi executat de către procesor. Restul timpului a fost ocupat cu rularea altor procese sau schimbarea contextului între procese. Schimbarea contextului are loc în momentul în care procesului îi expiră cuanta de timp sau accesează dispozitive periferice lente.

5.2.6 Sistemul de fișiere procfs

După cum s-a prezentat, sistemul de fișiere procfs este utilizat de comenzile ce obțin informații despre procesele din sistem: **ps**, **pstree**, **pgrep** sau **top**. Pachetul Debian/Ubuntu utilizat pentru instalarea acestor utilitare se numește `procps` pentru a indica faptul că procfs este utilizat pentru obținerea de informații despre procese.

procfs este un sistem de fișiere virtual (adică datele nu se găsesc pe un suport fizic permanent precum un hard disk sau CD-ROM). În cazul procfs, fiecare fișier este de fapt o regiune de memorie (stocată fizic în RAM) care oferă informații despre sistem. procfs oferă informații atât despre procesele din sistem cât și alte informații specifice: dispozitivele conectate, ocuparea întreruperilor etc, permițând totodată și configurarea anumitor componente ale sistemului.

procfs este montat în `/proc`:

```
razvan@asgard:~$ cat /etc/mtab
/dev/hda3 / ext3 rw,errors=remount-ro 0 0
tmpfs /lib/init/rw tmpfs rw,nosuid,mode=0755 0 0
proc /proc proc rw,noexec,nosuid,nodev 0 0
[...]
```

Din punctul de vedere al utilizatorului, procfs este vizibil ca o suită de fișiere și directoare cu informații despre sistem:

```
razvan@asgard:~$ cd /proc/

razvan@asgard:/proc$ ls
1/     2584/  2792/  2961/  3027/  97/          filesystems  mtrr
1021/  2585/  2793/  2963/  3028/  99/          fs/          net/
192/   2591/  2794/  2969/  3029/  acpi/        ide/         partitions
193/   2603/  2796/  2971/  3052/  asound/      interrupts   self
194/   2656/  2873/  2976/  3066/  buddyinfo    iomem        slabinfo
[...]
```

Informații despre procese

Informații despre procesele din sistem sunt oferite de procfs prin intermediul directoarelor al căror nume este un număr. Acest număr reprezintă pid-ul procesului despre care se dorește aflarea de informații.

În continuare vom prezenta diverse informații conținute în directorul din `/proc` asociat unui proces. Vom prezenta ca studiu de caz procesul folosit pentru editarea unui document Word. Fiind vorba de OpenOffice, numele executabilului din care a fost generat procesul este `soffice.bin`:

```
razvan@asgard:/proc$ ps -C soffice.bin
 PID TTY          TIME CMD
 3066 ?        00:01:02 soffice.bin

razvan@asgard:/proc$ cd 3066/

razvan@asgard:/proc/3066$ ls -F
auxv     environ  maps    mountstats  root    statm   wchan
cmdline  exe      mem     oom_adj     smaps   status
cwd      fd/      mounts  oom_score   stat    task/
```

Se poate observa că procesul asociat OpenOffice are pid-ul 3066 și ca atare va fi accesat directorul `3066`. Acest director conține o serie de fișiere, directoare și legături simbolice care oferă informații despre proces. Astfel, legătura simbolică `exe` oferă informații despre executabilul utilizat pentru crearea procesului; fișierul `cmdline` și fișierul `environ` precizează linia de comandă folosită și mediul de creare (environment):

```
razvan@asgard:/proc/3066$ ls -l exe
lrwxrwxrwx 1 razvan razvan 0 2007-08-04 12:01 exe -> /usr/lib/openoffice/
program/soffice.bin

razvan@asgard:/proc/3066$ cat cmdline
/usr/lib/openoffice/program/soffice.bin-writer-splash-pipe=5

razvan@asgard:/proc/3066$ cat environ
SSH_AGENT_PID=2950SHELL=/bin/bashGTK_RC_FILES=/etc/gtk/gtkrc:/home/razvan
/.gtkrc-1.2-gnome2OLDPWD=/usr/lib/openofficeUSER=
razvanOPENOFFICE_MOZILLA_FIVE_HOME=/usr/lib/openoffice/
programLD_LIBRARY_PATH=/usr/lib/openoffice/program
[...]
```

Fișierul `status` oferă, printre altele, informații referitoare la utilizatorul ce deține fișierul executabil (`uid 1000`), starea procesului (`S` – sleep), cantitatea de memorie ocupate, numărul de thread-uri etc:

```
razvan@asgard:/proc/3066$ cat status
Name:   soffice.bin
State:  S (sleeping)
SleepAVG:       98%
[...]
Uid:    1000    1000    1000    1000
Gid:    1000    1000    1000    1000
[...]
VmPeak:   205720 kB
VmSize:   202392 kB
VmLck:         0 kB
```

```
VmHWM:     84040 kB
VmRSS:     80968 kB
[...]
Threads:        6
[...]
```

Directorul `task/` oferă informații despre thread-urile acestui proces. Nu insistăm pe noțiunea de thread, deoarece depășește aria de cuprindere a acestui capitol.

```
razvan@asgard:/proc/3066$ cd task/

razvan@asgard:/proc/3066/task$ ls
3066  3067  3068  3069  3070  3071
```

Directorul `fd/` deține legături simbolice către fișiere deschise de procesul curent. Se observă, printre fișiere deschise, și fișierul cu extensia .doc care este editat (Raport.doc):

```
razvan@asgard:/proc/3066$ cd fd/

razvan@asgard:/proc/3066/fd$ ls
0  10  12  14  16  18  2   22  24  26  28  3   31  33  35  37  4  6  8
1  11  13  15  17  19  20  23  25  27  29  30  32  34  36  39  5  7  9

razvan@asgard:/proc/3066/fd$ ls -l
total 38
lr-x------ 1 razvan razvan 64 2009-08-04 13:31 0 -> /dev/null
l-wx------ 1 razvan razvan 64 2009-08-04 13:31 1 -> pipe:[8770]
l-wx------ 1 razvan razvan 64 2009-08-04 13:31 10 -> pipe:[9795]
[...]
lr-x------ 1 razvan razvan 64 2009-08-04 13:31 22 -> /dev/urandom
[...]
lrwx------ 1 razvan razvan 64 2009-08-04 13:31 34 -> /home/razvan/Raport.
doc
[...]
```

Alte informații utile

În afara informațiilor despre procese, procfs oferă și alte informații utile despre sistem. Acestea pot fi aflate tot prin accesarea unor fișiere și directoare din `/proc`. Printre informațiile utile (folosite, de asemenea, de diverse utilitare din sistem), se găsesc informații despre:

- procesor: prin accesarea `/proc/cpuinfo`;
- memoria sistemului: prin accesarea `/proc/meminfo`;
- partiții: prin accesarea `/proc/partitions`;
- timpul de rulare a sistemului: prin accesarea `/proc/uptime`;

În plus, accesarea directoarelor de forma `/proc/fs/`, `/proc/net/`, oferă informații suplimentare, permițând totodată configurarea unor parametri ai sistemului. Astfel procfs nu este numai o interfață de citire a informațiilor despre sistem, ci este și una de configurare a unor parametri ai sistemului.

5.3 Rularea proceselor în background. Job-uri. Daemoni

Rularea unui proces din interpretorul de comenzi (shell) rezultă de obicei în afișarea unui rezultat și terminarea procesului. Spre exemplu, dacă se dorește afișarea conținutului directorului curent se rulează comanda **`ls`**. Acest lucru duce la crearea unui proces fiu din shell, încărcarea executabilului `/bin/ls` și execuția noului proces încheiată cu terminarea acestuia. După terminarea procesului, utilizatorul poate rula un nou proces prin introducerea comenzii corespunzătoare la promptul interpretorului.

Un proces care rulează conform scenariului de mai sus se spune că rulează în **foreground** (în prim plan). Un proces care rulează în foreground are acces la terminalul curent. Altfel spus, procesele care rulează în foreground pot citi de la intrarea standard (standard input) și pot afișa informații la ieșirea standard (standard output).

De partea cealaltă, un proces poate rula în **background** (în fundal). Un proces care rulează în fundal pierde posibilitatea de a citi de la intrarea standard, dar își continuă rularea.

5.3.1 Rularea unui proces în background

Rularea unui proces în fundal se realizează cu ajutorul operatorului `&` după comandă. Acest operator va crea procesul asociat comenzii introduse și va forța rularea acestuia în fundal. Avantajul acestei abordări este faptul că se oferă înapoi promptul interpretorului către utilizator, în felul acesta utilizatorul putând introduce o nouă comandă. De obicei acest lucru se realizează când comanda de executat durează mult timp, ca de exemplu căutarea unui fișier, actualizarea bazei de date de căutare (**`updatedb`**), rularea unui proces cu interfață grafică etc. Procesele care rulează în fundal sunt denumite *job*-uri.

În exemplul de mai jos, se rulează comenzile de deschidere a editorului XEmacs și a calculatorului din mediul grafic GNOME. Ambele programe sunt pornite în fundal:

```
1 razvan@asgard:~$ xemacs &
2 [1] 3418
3
4 razvan@asgard:~$ gnome-calculator &
5 [2] 3420
6
7 razvan@asgard:~$
```

După rularea unei comenzi în fundal se observă că apare un mesaj specific de forma `[N] M`. În acest format `N` este indicele *job*-ului (indicele procesului care rulează în fundal), iar `M` este identificatorul de proces (pid-ul) pentru *job*-ul din fundal. Avantajul rulării acestor procese în fundal, așa cum a fost precizat și mai sus, este faptul că interpretorul de comenzi oferă înapoi promptul utilizatorului permițându-i astfel rularea de noi comenzi.

5.3.2 Suspendarea unui proces

Un proces care rulează în *foreground* poate fi forțat să intre în *background* (și implicit să intre în starea suspendat – nu rulează) prin intermediul combinației de taste `CTRL-Z` (control ținut apăsat, după care se apasă z). Apăsarea acestei combinații de taste este interceptată de sistemul de operare și procesul activ (aflat în *foreground*) este forțat să fie suspendat.

În exemplul de mai jos, utilizatorul a dorit editarea fișierului `out.txt` folosind *vi*. Procesul creat prin execuția comenzii **`vi out.txt`** rulează inițial în *foreground*. Utilizatorul a apăsat apoi `CTRL-Z` și a forțat suspendarea procesului curent în *background* și oferirea promptului către utilizator:

```
1  razvan@asgard:~$ vi out.txt
2  [3]+  Stopped                 vi out.txt
3
4  razvan@asgard:~$
```

Se observă că procesul a fost suspendat (*stopped*) și s-a creat jobul cu numărul 3.

Un scenariu asemănător este prezentat și în exemplul de mai jos. De data aceasta utilizatorul folosește utilitarul **`du`** pentru a vedea spațiul ocupat de fișierele și directoarele din directorul curent:

```
1  razvan@asgard:~$ du -hs *
2  1022M   Desktop
3  12K     Download
4  528K    bin
5  24M     books
6  756K    code
7  532K    junk
8  3.2M    official
9  4.0K    out.txt
10 29M     packages
11 165M    people
12 617M    pictures
13 116M    projects
14
15 [4]+  Stopped                 du -hs *
16
17 razvan@asgard:~$
```

5.3.3 Controlul job-urilor

Termenul de *job* a fost introdus referitor la procesele care rulează sau sunt suspendate în fundal. Interpretorul de comenzi (shell-ul) bash pune la dispoziția utilizatorului mai multe comenzi prin intermediul cărora se poate interacționa cu procesele din fundal. Mai multe informații despre aceste comenzi pot fi găsite prin consultarea paginii de manual (**`man bash`**, secțiunea `JOB CONTROL`) sau pagina info (**`info bash`**, secțiunea "JOB CONTROL". Trebuie instalat pachetul `bash-doc`).

Comanda jobs

Comanda **jobs** afișează *job*-urile pentru terminalul curent:

```
razvan@asgard:~$ jobs
[1]   Running                 xemacs &
[2]   Running                 gnome-calculator &
[3]-  Stopped                 vi out.txt
[4]+  Stopped                 du -hs *

razvan@asgard:~$
```

Se poate observa că sunt patru *job*-uri dintre care două sunt în rulare (*Running*) iar celelalte două sunt suspendate (*Stopped*). Semnul + este folosit pentru *job*-ul curent (cel mai recent job). Semnul - este folosit pentru *job*-ul anterior *job*-ului curent.

Comenzile bg și fg

Utilizatorul poate determina rularea comenzilor care sunt suspendate (fie au fost pornite în fundal, fie au fost suspendate prin combinația de taste CTRL-Z). Aceste procese își pot continua execuția în *foreground* sau în *background*. Daca se dorește continuarea în *background* a execuției unui job suspendat, se folosește comanda **bg**; daca se dorește continuarea execuției în *foreground* a unui job suspendat, se folosește comanda **fg**.

În exemplul de mai jos, utilizatorul a decis continuarea execuției job-ului [4] în background. Rularea comenzii **bg** fără niciun parametru are ca efect continuarea job-ului curent. Daca se dorește rularea job-ului n, se folosește argumentul %n (spre exemplu **bg %5** pentru rularea în background a job-ului cu indicele 5).

```
razvan@asgard:~$ bg %4
[4]+ du -hs * &
1.2G    school
2.5M    shared-projects
1.7G    vmware

razvan@asgard:~$ jobs
[1]   Running                 xemacs &
[2]   Running                 gnome-calculator &
[3]+  Stopped                 vi out.txt
[4]-  Done                    du -hs *

razvan@asgard:~$ jobs
[1]   Running                 xemacs &
[2]-  Running                 gnome-calculator &
[3]+  Stopped
```

Se poate observa că s-a continuat rularea job-ului [4] în background după care procesul asociat s-a terminat.

Mesajul Done este afișat la încheierea rulării unor job-uri. Spre exemplu, daca vom închide din mediul grafic gnome-calculator și xemacs, mesajele afișate vor fi asemănătoare cu cele de mai jos:

```
razvan@asgard:~$
[1]   Done                    xemacs
[2]-  Done                    gnome-calculator
```

```
4
5 razvan@asgard:~$ jobs
6 [3]+  Stopped                 vi out.txt
```

După terminarea celor două job-uri, mai rămâne în background doar job-ul 3 (cel asociat editorului `vi`). Având în vedere că este singurul proces rămas, `vi` poate fi adus în foreground utilizând comanda **`fg`** sau **`fg %3`**:

```
1 razvan@asgard:~$ fg
2 vi out.txt
```

După rularea acestei comenzi, utilizatorul poate continua editarea fișierului `out.txt`.

5.3.4 Daemoni

> Un **daemon** este un tip particular de proces care rulează în fundal. Față de procesele care rulează în fundal (pornite prin intermediul operatorului &) sau care sunt suspendate în fundal (prin intermediul combinației `CTRL-Z`), un daemon se detașează de terminalul de control.

Astfel, un daemon nu va putea comunica direct cu utilizatorul prin intermediul terminalului: nu va accepta comenzi de control de la tastatură și nu va afișa rezultate la terminal. Comunicarea cu un daemon se face prin mecanisme mai complicate (ca de exemplu semnale – vezi secțiunea 5.4).

Inițial denumirea de daemon nu a avut nicio semnificație. Ulterior a fost găsită o abreviere pentru aceasta (*backronym*) de la *Disk And Execution MONitor*.

De obicei, procesele care sunt daemoni au numele terminat în d: `hald`, `udevd`, `sshd`, `inetd` etc. (fără a fi însă o regulă).

Pentru selectarea proceselor care nu au atașate niciun terminal (și ca atare pot fi daemoni), se folosește **`ps`** cu argumentul `-t` (pentru selecția după terminal) urmat de opțiunea - (semnul minus înseamnă procesele care nu au atașate niciun terminal).

```
1  razvan@asgard:~$ ps -t -
2    PID TTY          TIME CMD
3      1 ?        00:00:00 init
4      2 ?        00:00:00 ksoftirqd/0
5      3 ?        00:00:00 events/0
6  [..]
7   2657 ?        00:00:00 exim4
8   2665 ?        00:00:00 fbguard
9   2666 ?        00:00:00 fbserver
10  2676 ?        00:00:00 inetd
11  2689 ?        00:00:00 sshd
12  2695 ?        00:00:00 vsftpd
13 [...]
14  3006 ?        00:00:02 gedit
15  3009 ?        00:00:00 gnome-screensav
16  3016 ?        00:00:00 gnome-terminal
17  3017 ?        00:00:00 gnome-pty-helper
```

Se observă că nu toate procesele care nu au atașate un terminal sunt procese daemon. Spre exemplu, procesul gedit (pid 3006) nu este un daemon în adevăratul

sens al cuvântului. Deși nu are atașat niciun terminal de control, se poate comunica prin intermediul mediului grafic. În plus față de a nu avea asociat niciun terminal, un proces daemon va rula în background, eliminând posibilitatea comunicării directe (tastatură/mouse) cu utilizatorul.

Interacțiunea cu procesele daemon

Se pune însă întrebarea cum poate utilizatorul să interacționeze cu procesele daemon, în măsura în care nu o poate face prin intermediul intrării și ieșirii standard. Cele două mecanisme care permit acest lucru sunt fișierul/fișierele de configurare și semnalele.

În general, procesele daemon au unul sau mai multe fișiere de configurare. De exemplu:

- server-ul web **apache2** folosește `/etc/apache2/apache2.conf`;
- server-ul SSH **openssh** folosește `/etc/ssh/sshd_config`;
- server-ul **X** (mediul grafic) folosește `/etc/X11/xorg.conf`;

Aceste fișiere de configurare sunt citite în momentul creării procesului daemon și permit controlul funcționării acestuia. De exemplu, în cazul server-ului openssh, fișierul de configurare permite controlul tipului de autentificare (cu parolă sau prin chei publice), portul pe care serverul ascultă, dacă serverul permite X forwarding etc.

Dacă se dorește alterarea comportamentului server-ului, se editează fișierul de configurare. După salvarea fișierului, trebuie repornit procesul daemon, acesta va citi fișierul de configurare și va stabili noul comportament conform configurației din fișier.

Oprirea, suspendarea și repornirea unui proces, în general, și a unui daemon, în particular, se realizează cu ajutorul semnalelor. Daca în cazul unui proces care nu este daemon, acesta poate fi oprit sau suspendat și prin intermediul terminalului (prin combinații de taste), un proces daemon poate fi oprit/suspendat/repornit doar cu ajutorul semnalelor. Despre semnale se va discuta în secțiunea 5.4.

Comanda nohup

Comanda **nohup** poate fi utilizată pentru a rula un proces cu caracteristici de daemon. Utilizarea acestei comenzi determină ignorarea de către proces a semnalului SIGHUP (acest semnal este trimis de un terminal către procesele sale copil atunci când este închis). **nohup** este folosit, de obicei, atunci când utilizatorul va dori să părăsească terminalul dar comanda să fie rulată în continuare. Comanda **nohup** este urmată de comanda ce va fi rulată pentru pornirea procesului și de caracterul `&`. Un exemplu este prezentat în continuare:

```
1 asgard:/home/razvan# nohup updatedb &
2 [1] 3116
3
4 asgard:/home/razvan# nohup: appending output to 'nohup.out'
5
6 asgard:/home/razvan# ps -e
7 [...]
8  3116 pts/1    00:00:00 updatedb
9  3124 pts/1    00:00:00 updatedb
```

```
10  3125 pts/1    00:00:00 find
11  3126 pts/1    00:00:00 sort
12  3127 pts/1    00:00:00 frcode
13  3129 pts/1    00:00:00 ps
```

Se observă că procesul creat (`updatedb`) nu pierde controlul terminalului curent, însă nu se poate comunica cu acesta prin intermediul semnalelor de suspendare/repornire sau intrării/ieșirii standard. După cum reiese și din rezultatul rulării comenzii, ieșirea acesteia este redirecționată în fișierul `nohup.out`.

5.4 Semnale

Semnalele sunt mecanisme de notificare asincronă care sunt utilizate pentru a transmite o condiție specială unui proces.

Semnalele sunt asincrone în raport cu fluxul de rulare al procesului. Astfel, un semnal nu este corelat direct cu instrucțiunile executate de proces, putând fi transmis/primit la un moment de timp neștiut de proces.

Semnalele sunt transmise în două moduri:

- **de nucleu** (kernel) pentru a indica o condiție neobișnuită care solicită oprirea/suspendarea sau doar notificarea procesului;
- **de utilizator**.

Semnale care sunt transmise uzual de nucleu sunt următoarele (aceste semnale se pot trimite și de către utilizator, dar cel mai adesea tunt trimise de către nucleu):

- `SIGSEGV` (*signal segment violation*): în momentul accesării invalide a unei zone de memorie;
- `SIGBUS` în momentul unei erori pe magistrala sistemului;
- `SIGFPE` (*signal floating point error*): în momentul apariției unei erori de virgulă mobilă;
- `SIGTERM` (*signal termination*): pentru oprirea unui proces;
- `SIGKILL` (*signal kill*): pentru oprirea necondiționată a unui proces.

Semnale care sunt transmise uzual de utilizator sunt:

- `SIGINT` (*signal interrupt*): întrerupe procesul curent;
- `SIGQUIT` (*signal quit*): semnalează oprirea procesului curent;
- `SIGSTOP` (*signal stop*): suspenda procesul curent;
- `SIGCONT` (*signal continue*): repornește procesul suspendat.

Nu vom insista pe modul și condițiile în care nucleul transmite semnale unui proces, ci pe modul în care utilizatorul le transmite. Utilizatorul poate transmite unui proces semnale în două moduri:

- prin intermediul unor comenzi specifice: **`kill`**, **`killall`**; (vezi secțiunea 5.4.2)

- prin intermediul unor combinații de taste specifice. (vezi secțiunea 5.4.3)

Informații detaliate despre semnale se pot afla prin consultarea paginii de manual (**`man 7 signal`**).

5.4.1 Semnale importante UNIX

Înainte de a discuta despre modul de interacțiune a utilizatorului cu procesele prin intermediul semnalelor, trebuiesc amintite câteva semnale importante (unele menționate și anterior). Lista completă cu semnalele pe care le oferă sistemul de operare poate fi vizualizată folosind comanda **`kill`** cu opțiunea `-l` (*list*):

```
razvan@asgard:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Se observă că un semnal are un număr de identificare și un nume care oferă indicații despre efectul său asupra proceselor. Semnalele relevante sunt cele până la 31 (`SIGSYS`). Celelalte (`SIGRTMIN`, `SIGRTMAX`) sunt semnale de timp real (*SIGnal Real Time*) și nu sunt relevante pentru prezentarea curentă. Vom prezenta câteva dintre semnalele cele mai importante:

- `SIGHUP` (1) (*signal hang up*): este folosit pentru repornirea unui proces; acest semnal este de obicei transmis proceselor daemon pentru repornirea acestora și recitirea fișierului/fișierelor de configurare;
- `SIGINT` (2) (*signal interrupt*): este folosit pentru a întrerupe un proces; de obicei, efectul imediat este terminarea procesului;
- `SIGQUIT` (3) (*signal quit*): este folosit pentru închiderea unui proces; este mai puternic decât `SIGINT`: unele procese pot fi terminate cu `SIGQUIT` dar nu cu `SIGINT`;
- `SIGILL` (4) (*signal illegal*): procesul execută o instrucțiune invalidă;
- `SIGKILL` (9) (*signal kill*): cel mai puternic semnal; termină în mod necondiționat un proces
- `SIGSEGV` (11) (*signal segment violation*): semnal transmis în momentul accesului invalid la o zona de memorie; rezultă, de obicei, în terminarea procesului și la apariția mesajului “Segmentation fault”;

- `SIGTERM` (15) (*signal termination*): termină un proces; în mod tipic nucleul transmite întâi `SIGTERM` unui proces pentru a-l anunța că va fi terminat; procesul realizează operații de curățare (*cleanup*), după care kernel-ul transmite `SIGKILL` și procesul este terminat;
- `SIGSTOP` (19) (*signal stop*): suspendă procesul curent;
- `SIGCONT` (18) (*signal continue*): reia execuția unui proces suspendat.

5.4.2 Comenzile kill, killall și pkill

Comenzile **`kill`**, **`killall`** și **`pkill`** sunt utilizate pentru a transmite un semnal unui proces sau unui set de procese.

Comanda kill

Comanda **`kill`** transmite un semnal unui proces. Comanda primește ca argument pid-ul procesului (sau pid-urile proceselor) către care se dorește să se transmită semnalul. În mod implicit comanda transmite semnalul `SIGTERM`:

```
1  razvan@asgard:~$ xcalc &
2  [1] 3590
3
4  razvan@asgard:~$ ps
5    PID TTY          TIME CMD
6   3092 pts/1    00:00:00 bash
7   3590 pts/1    00:00:00 xcalc
8   3591 pts/1    00:00:00 ps
9
10 razvan@asgard:~$ kill 3590
11
12 razvan@asgard:~$
13 [1]+  Terminated              xcalc
14
15 razvan@asgard:~$ ps
16   PID TTY          TIME CMD
17  3092 pts/1    00:00:00 bash
18  3592 pts/1    00:00:00 ps
```

Comanda poate fi configurată pentru a transmite și alt semnal în afară de `SIGTERM`. Astfel, dacă se dorește transmiterea semnalului `SIGSTOP` către procesele cu pid-urile 1234 și 5678, se poate utiliza una din alternativele:

```
1  razvan@asgard:~$ kill -SIGSTOP 1234 5678
2
3  razvan@asgard:~$ kill -STOP 1234 5678
4
5  razvan@asgard:~$ kill -19 1234 5678
```

Se observă că semnalul poate fi transmis ca parametru fie în formă literală (`SIGSTOP`, `STOP`), fie în formă numerică (19).

Comanda killall

Comanda **killall** este folosită pentru transmiterea unui semnal către un proces sau către un set de procese atunci când se cunoaște numele procesului. Un exemplu de utilizare este prezentat mai jos:

```
root@asgard:/home/razvan# ps -C apache2 -H
  PID TTY          TIME CMD
 2792 ?        00:00:00 apache2
 2793 ?        00:00:00   apache2
 2794 ?        00:00:00   apache2
 2796 ?        00:00:00   apache2

root@asgard:/home/razvan# killall -KILL apache2

root@asgard:/home/razvan# ps -C apache2
  PID TTY          TIME CMD
```

În acest exemplu, s-a trimis semnalul de terminare a unui proces (`SIGKILL`) către toate procesele apache2 din sistem. Modul de transmitere a semnalului ca argument al comenzii este același ca în cazul comenzii **kill**.

O opțiune care poate fi utilă este opțiunea `--user` care permite transmiterea unui semnal către toate procesele unui anumit utilizator. Pentru a putea utiliza această opțiune, utilizatorul care apelează **killall** trebuie să fie `root`:

```
root@asgard:/home/razvan# ps -o pid,comm,user -u guest
  PID COMMAND         USER
 3761 su              guest
 3762 bash            guest
 3802 su              guest
 3803 bash            guest
 3822 top             guest
 3825 bc              guest

asgard:/home/razvan# killall --user guest

asgard:/home/razvan# ps -o pid,comm,user -u guest
  PID COMMAND         USER
```

În situația de mai sus s-a transmis semnalul implicit (`SIGTERM`) către toate procesele utilizatorului guest, rezultând în terminarea acelor procese.

Comanda pkill

Comanda **pkill** poate fi comparată cu comanda **pgrep**. În timp ce comanda **pgrep** oferea informații despre procesele care îndeplineau anumite criterii, comanda **pkill** transmite un semnal proceselor pe baza unor criterii de selecție a acestora.

La fel ca în cazul **pgrep**, selecția se poate realiza după pid-ul procesului părinte, după terminalul utilizat, după id-ul utilizatorului sau al grupului. Argumentul semnal se transmite la fel ca în cazul comenzilor **kill** și **killall**.

5.4.3 Transmiterea de semnale prin combinații de taste

În afara comenzilor de mai sus, un utilizator poate transmite anumite semnale procesului curent (cel care rulează în acel moment în terminal) prin intermediul unor combinații de taste specifice.

Astfel de combinații sunt:

- `CTRL-Z`: transmite semnalul `SIGSTOP` către procesul curent, care are drept consecință suspendarea acestuia. O situație utilă este atunci când se rulează editorul *vi* și se dorește rularea altei comenzi. Se folosește `CTRL-Z` pentru suspendarea procesului vi și revenirea la promptul terminalului, unde poate fi introdusă noua comandă. După introducerea comenzii se folosește comanda **`fg`** pentru a reactiva procesul vi.
- `CTRL-C`: transmite semnalul `SIGINT` către procesul curent, care, în general, va omorî procesul. De obicei, combinația de taste se folosește în cazul în care un proces este blocat.
- `CTRL-\`: transmite semnalul `SIGQUIT`. Acesta are același rol ca și `CTRL-C`, doar că este mai puternic.

Trebuie precizat că nu toate aceste combinații de taste au efect pentru toate procesele, întrucât unele procese pot **ignora semnalele** transmise (pot avea un comportament non-standard la primirea unui semnal).

5.5 Comunicația între procese

La fel cum în cadrul unui departament mai mulți oameni conlucrează pentru a îndeplini o sarcină, și în cadrul unui sistem de operare procesele acestuia interacționează. Interacțiunea proceselor într-un sistem de operare poartă numele de **comunicație între procese**.

De ce este nevoie de comunicație între procese? În primul rând pentru ca un proces poate avea nevoie de resursele pe care i le pune la dispoziție un alt proces. Spre exemplu, în cazul utilitarului **`top`**, acesta culege informații pe care i le furnizează alte procese prin intermediul nucleului sistemului de operare și al sistemului de fișiere procfs.

În altă situație, un proces A trebuie să aștepte ca un alt proces B să termine o sarcină pentru ca A să poată continua. Aceasta formă de comunicație între A și B se numește sincronizare: un proces trebuie să aștepte un alt proces.

Comunicația între procese este intermediată în diverse moduri: cu ajutorul unui fișier, a unui socket, a unei zone de memorie sau a unui canal de comunicație (pipe). Intermedierea printr-un fișier este destul de simplă: un proces scrie o informație într-un fișier, iar un alt proces o citește. Nu vom prezenta comunicația prin intermediul unui socket sau a unei regiuni de memorie, întrucât depășesc aria de cuprindere a acestui capitol. Vom discuta, însă, despre comunicația prin intermediul unui canal de comunicație (pipe).

5.5.1 Operatorul | (pipe)

Operatorul | (**pipe**) este folosit pentru a asigura comunicația între două procese rulate din linia de comandă, folosind un canal de comunicație numit *pipe*.

Comunicația prin intermediul unui pipe este o îmbunătățire a comunicației prin intermediul unui fișier. Astfel, în cazul comunicației prin intermediul unui fișier, un proces scrie datele de ieșire într-un fișier iar un alt proces folosește acel fișier ca intrare. În cazul comunicației prin pipe, ieșirea primului proces este folosită direct ca intrare pentru al doilea, fără a mai fi nevoie de un fișier pe disc.

Vom folosi pentru exemplificare comanda **`grep`**, care caută un cuvânt în cadrul unui fișier. Dacă, spre exemplu, dorim aflarea de informații despre procesele bash din sistem folosind **`grep`**, ar trebui, în primă fază, să redirectăm ieșirea comenzii **`ps`** într-un fișier, iar apoi să folosim **`grep`** pe acel fișier, ca în exemplul de mai jos:

```
1  razvan@asgard:~$ ps -e > ps.out
2
3  razvan@asgard:~$ grep bash ps.out
4   3029 pts/0    00:00:00 bash
5   3092 pts/1    00:00:00 bash
6   3299 pts/2    00:00:00 bash
7   3317 pts/2    00:00:00 bash
8   3740 pts/3    00:00:00 bash
9   3758 pts/3    00:00:00 bash
10  3781 pts/4    00:00:00 bash
11  3799 pts/4    00:00:00 bash
```

Același rezultat ca mai sus poate fi însă realizat prin intermediul operatorului |, fără a mai fi nevoie de intermedierea printr-un fișier:

```
1  razvan@asgard:~$ ps -e | grep bash
2   3029 pts/0    00:00:00 bash
3   3092 pts/1    00:00:00 bash
4   3299 pts/2    00:00:00 bash
5   3317 pts/2    00:00:00 bash
6   3740 pts/3    00:00:00 bash
7   3758 pts/3    00:00:00 bash
8   3781 pts/4    00:00:00 bash
9   3799 pts/4    00:00:00 bash
```

Pentru a exemplifica puterea acestui operator vom considera următoarea problemă: dorim să aflăm primii 3 utilizatori din sistem care au directorul de bază în `/home`, ordonați în ordinea alfabetică a numelui de utilizator.

Informații despre utilizatorii din sistem se găsesc în `/etc/passwd`. Vom folosi grep pentru a extrage utilizatorii care au directorul de bază în `/home`:

```
1  razvan@asgard:~$ cat /etc/passwd | grep /home
2  razvan:x:1000:1000:razvan,,,:/home/razvan:/bin/bash
3  haldaemon:x:105:106:Hardware abstraction layer,,,:/home/haldaemon:/bin/
   false
4  guest:x:1001:1001:Guest Account,,,:/home/guest:/bin/bash
5  ftp:x:107:65534::/home/ftp:/bin/false
```

În continuare vom sorta rezultatul obținut, după care vom reține numai primele 3 intrări:

```
1  razvan@asgard:~$ cat /etc/passwd | grep /home | sort
```

```
ftp:x:107:65534::/home/ftp:/bin/false
guest:x:1001:1001:Guest Account,,,:/home/guest:/bin/bash
haldaemon:x:105:106:Hardware abstraction layer,,,:/home/haldaemon:/bin/
false
razvan:x:1000:1000:razvan,,,:/home/razvan:/bin/bash

razvan@asgard:~$ cat /etc/passwd | grep /home | sort | head -3
ftp:x:107:65534::/home/ftp:/bin/false
guest:x:1001:1001:Guest Account,,,:/home/guest:/bin/bash
haldaemon:x:105:106:Hardware abstraction layer,,,:/home/haldaemon:/bin/
false
```

5.6 Swapping

Un concept important în studiul sistemelor de operare și al proceselor acestora este conceptul de **swapping**. Swapping este o măsură compensare a cantității insuficiente de memorie RAM de care dispune un sistem la un moment dat. Pentru compensare, sistemul de operare va folosi o porțiune din disc pentru stocarea datelor din RAM.

În cazul unui sistem încărcat (cu multe procese), memoria RAM se poate dovedi insuficientă pentru a satisface toate procesele. În această situație o parte din paginile de memorie din RAM sunt evacuate (*swapped*) pe disc pentru a face loc pentru paginile de memorie utile în momentul de față. În momentul în care paginile evacuate sunt din nou necesare, se vor evacua alte pagini pentru a se aduce la loc cele curente.

Principiul swapping-ului este prezentat în figura de mai jos:

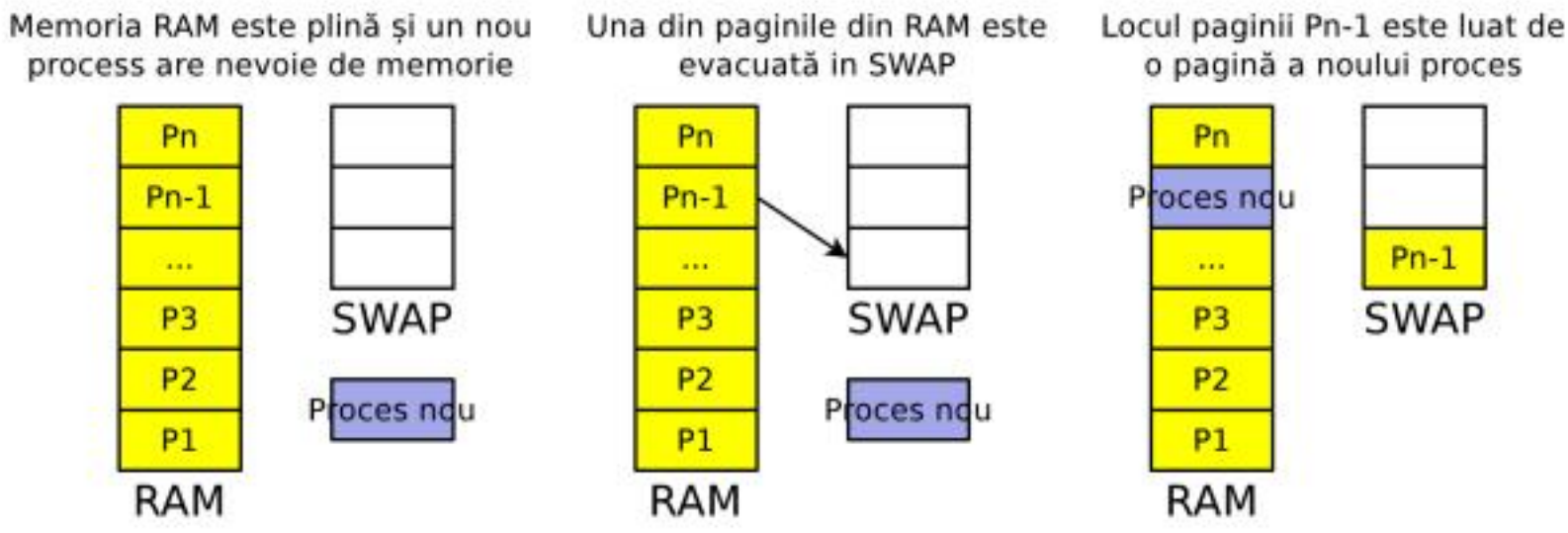


Figura 5.3: Swaping

Pe sistemele Linux swapping-ul este realizat în general pe partiții dedicate (partiția de swap – vezi secțiunea 2.2.2), care sunt folosite în cazul în care memoria din sistem este insuficientă. În cazul Windows, spațiul de swap alocat din partițiile sistemului, fiind utilizate în acest scop fișiere. Pentru a afla informații despre spațiul de swap utilizat în Linux se poate folosi comanda **`free`** (aceste informații se găsesc și în zona de sumarizare din ecranul `top`):

```
razvan@asgard:~$ free
             total       used       free     shared    buffers     cached
Mem:        514748     336876     177872          0      19892     185200
-/+ buffers/cache:     131784     382964
Swap:       497972          0     497972
```

Se observă că sistemul beneficiază de 512 MB de memorie RAM și nu este folosit spațiul de swap.

5.7 Studii de caz

5.7.1 Managementul proceselor/serviciilor pe Windows

În Windows, un proces are caracteristici asemănătoare cu procesele din Linux. Totuși, deși un proces este creat prin intermediul altui proces, nu există o ierarhie de procese și nici noțiunea directă de proces părinte și proces fiu. La fel ca în Linux, un proces este identificat printr-un PID, are asociate zone de memorie și fișiere deschise etc.

Task manager

Interfața de vizualizare și gestiune a proceselor în Windows este asigurată de Task Manager. Pentru pornirea acestuia se folosește combinația de taste `CTRL-ALT-DEL` sau `CTRL-SHIFT-ESC` sau click dreapta pe bara de task-uri și selectarea opțiunii `Task Manager` din meniu.

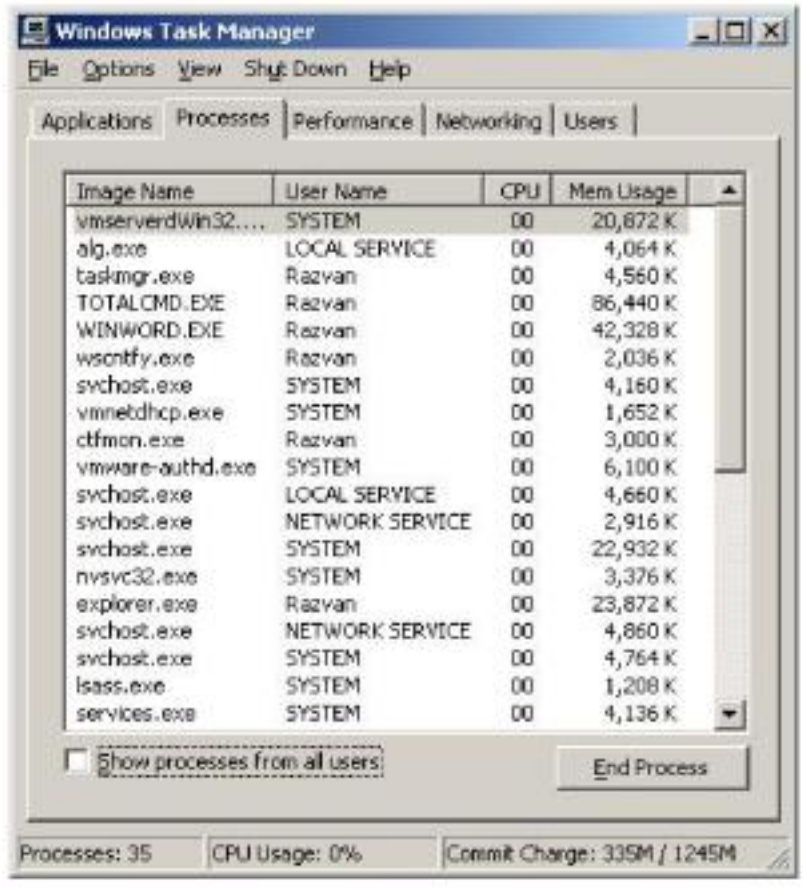


Figura 5.4: Windows Task Manager

Task Manager poate fi comparat cu utilitarul top. Afișează informații despre procese și despre sistem și poate fi configurat pentru personalizarea afișării sau pentru interacțiunea cu procesele din sistem.

În mod implicit, Task Manager afișează numele programului care a generat procesul, utilizatorul care a creat procesul, procentajul de procesor folosit și memoria utilizată. Dacă se dorește afișarea altor câmpuri se poate accesa meniul `View->Select`

`Columns`. Se pot afișa PID-ul procesului, memoria virtuală utilizată, maximul de memorie utilizat, numărul de fișiere deschise, numărul de thread-uri etc.

În afara vizualizării proceselor din sistem, Task Manager mai oferă următoarele servicii:

- permite crearea unui nou proces prin introducerea imaginii de executabil asociate (`File->New Task (Run...)`);
- permite monitorizarea și analiza performanței sistemului, prin accesarea tab-ului `Performance`; informațiile de aici sunt asemănătoare cu cele oferite de top în zona de sumarizare;
- permite deconectarea unui utilizator, sau oprirea, repornirea sau suspendarea sistemului (meniul `Shut Down`);
- permite terminarea unui proces; se selectează procesul dorit și se apasă butonul `End Process`.

Se poate observa că Windows Task Manager oferă cea mai mare parte din facilitățile furnizate de comenzile Unix de interacțiune cu procese sistemului. Un utilitar puternic de vizualizare și control a proceselor pe Windows este ProcessExplorer[1].

Services

În Windows, procesele daemon poartă numele de **servicii**. Ca și în Linux, serviciile sunt procese care rulează în *background* și sunt folosite pentru monitorizarea și întreținerea sistemului.

Fereastra de interacțiune cu serviciile Windows este afișată, în Windows XP, prin accesarea `Start->Control Panel->Administrative Tools->Services` sau click dreapta pe `My Computer->Manage->Services`.

Ecranul afișat prezintă toate serviciile sistemului. Coloanale afișate indică numele serviciului, o scurtă descriere, starea lui (oprit sau pornit), modul de pornire a serviciului (dezactivat, manual sau automat și, la Windows Vista, întarziat). Pentru pornirea, oprirea sau repornirea serviciului, ca și pentru schimbarea modului de pornire, se accesează meniul contextual al serviciului (click dreapta) și se selectează opțiunea `Properties`:

5.7.2 Procese importante

Orice sistem de operare deține un set de procese cu rol important în rularea și întreținerea sistemului de operare.

Procese importante Linux

Procese importante în Linux sunt:

[1] http://www.microsoft.com/technet/sysinternals/utilities/ProcessExplorer.mspx

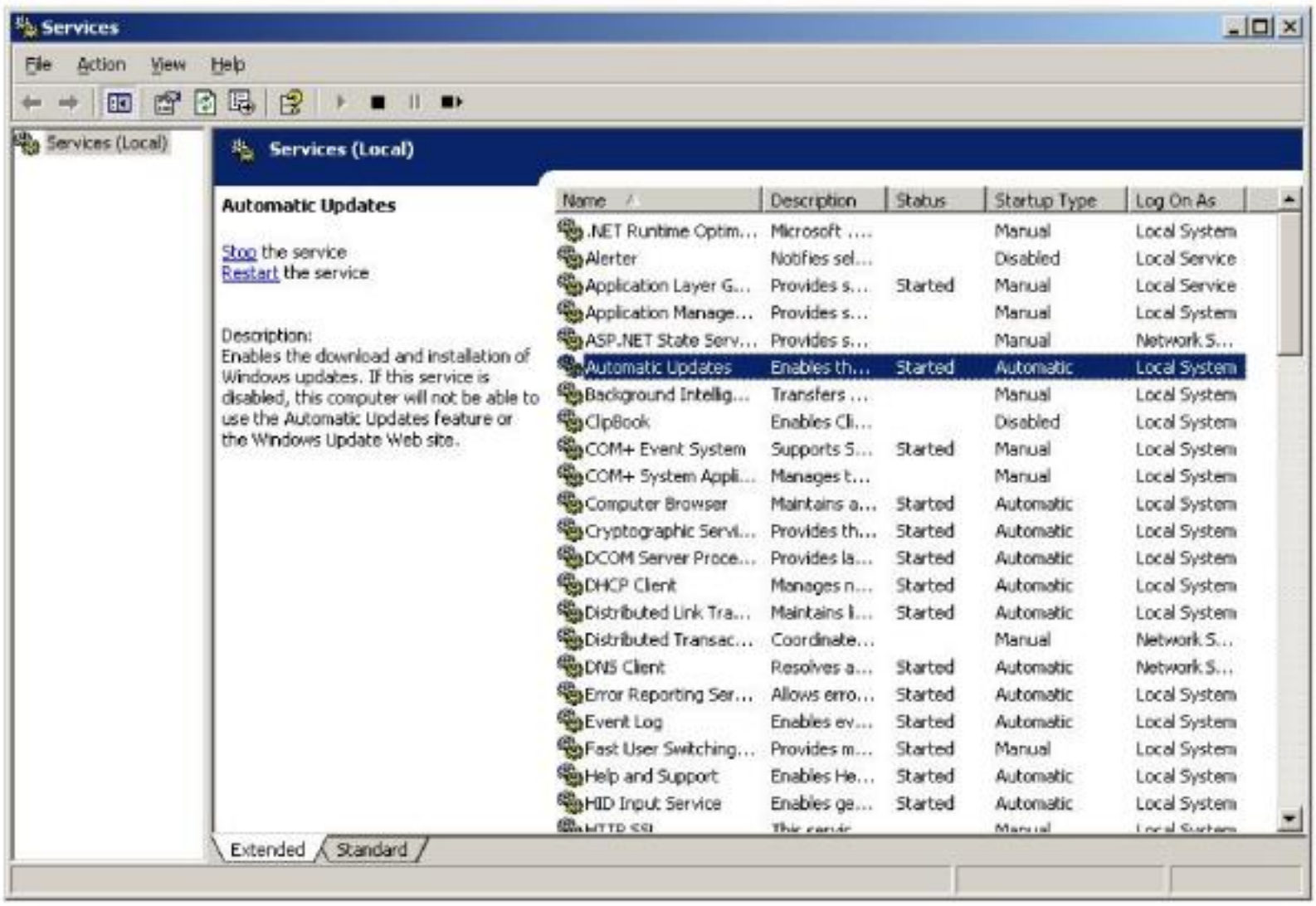


Figura 5.5: Windows Services

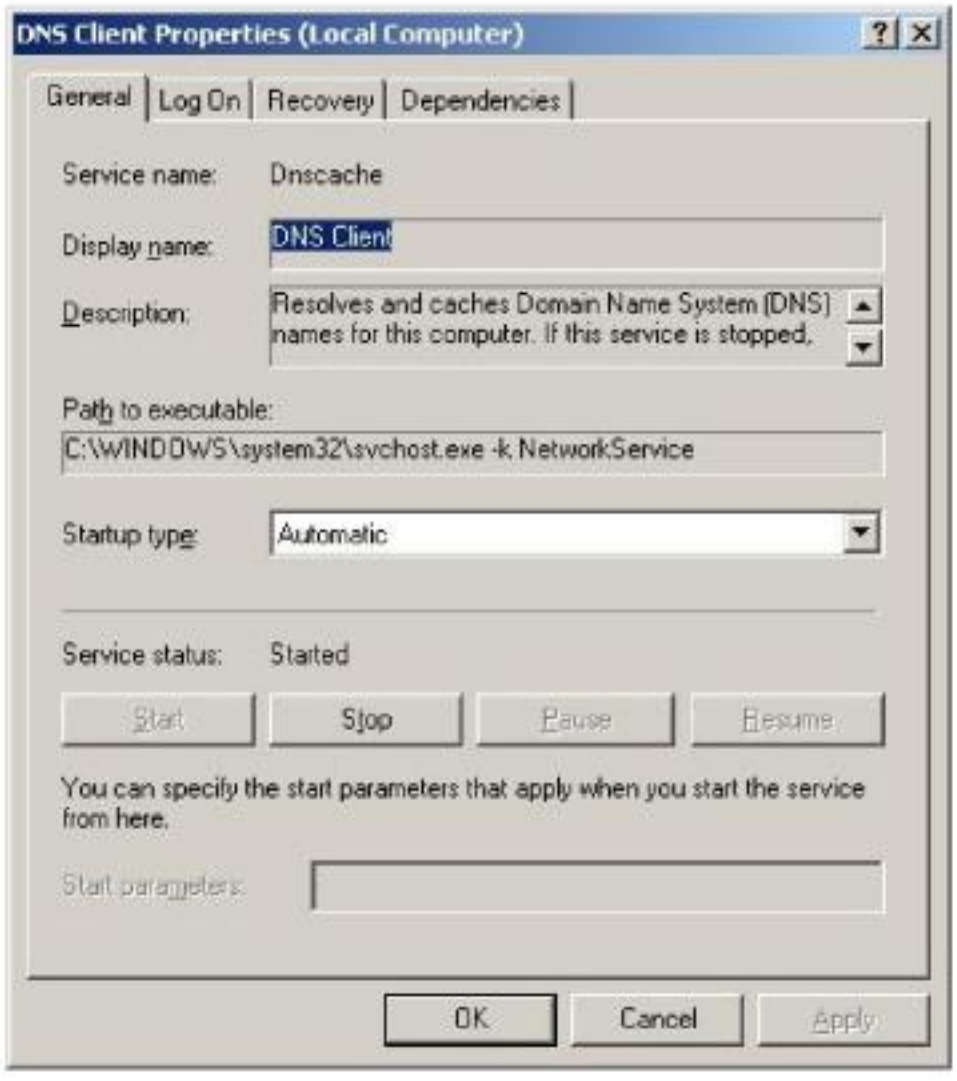


Figura 5.6: Interfața de control a serviciului Windows de DNS

- `init`: este procesul cu pid-ul 1 în sistemele Unix și este părintele tuturor proceselor; rolul său nu se limitează doar la a porni celelalte procese importante

în sistem ci și de a deveni procesul părinte pentru procesele rămase orfane;

- `kswapd` (kernel swap daemon): este responsabil cu alegerea paginilor de memorie care vor fi evacuate pe disc în momentul în care memoria este insuficientă;
- `pdflush` (*page daemon*): este responsabil cu asigurarea consistenței buffer-elor folosite de sitemul de fișiere;
- `getty`: este procesul resposabil cu autentificarea în sistem în terminalele virtuale; getty oferă promptul login : utilizatorului;
- `Xorg`: este procesul resposabil cu pornirea interfeței grafice și are asociat terminalul virtual tty7 (ce poate fi accesat prin combinația `ALT+CTRL-F7`).

Procese importante Windows

În Windows procese importante sunt:

- `lsass` (*Local Security and Authentication Server*): verifică validitatea autentificării utilizatorilor; în cazul unei autentificări corecte generează un jeton folosit pentru deschiderea unei prime sesiuni shell pe sistem; lsass este și numele unui virus;
- `svchost` (*Generic Host Process for Win32 Services*): este procesul folosit pentru gestiunea serviciilor din Windows;
- `csrss` (*Client/Server Run-time Subsystem*): este responsabil pentru crearea ferestrelor de consolă și gestiunea thread-urilor;
- `winlogon`: este o componentă esențială a subsistemului de autentificare în Windows;
- `explorer`: este este shell-ul unui sistem Windows, însă cu interfață grafică.

Mai multe informații despre procesele importante Windows găsiți la[1].

5.7.3 Prioritatea unui proces

În sistemele de operare moderne, procesele au asociată o **prioritate**. Aceasta influențează probababilitatea ca un proces să fie planificat pe procesor: cu cât un proces are o prioritate mai mare, cu atât probabilitatea ca el să fie următorul proces planificat pentru execuție crește. De obicei procesele care se blochează des (denumite I/O intensive) au asociate o prioritate mai mare, în timp ce procesele care utilizează intens procesorul (CPU intensive), au o prioritate mai mică.

Prioritatea proceselor nu este fixă. Sistemul de operare poate modifica prioritatea proceselor pentru a asigura echitatea planificării. Totodată, și utilizatorul poate modifica prioritatea unui proces dacă dorește acest lucru.

[1] http://www.neuber.com/taskmanager/process/

Schimbarea priorității unui proces în Linux

În Linux, prioritatea unui proces poartă numele de **niceness** (cât de echitabil se comportă un proces raportat la celelalte). Un proces are o prioritate mai mare cu cât valoarea sa nice este mai mică. Valorea nice pentru un proces se încadrează în limitele [-20, 19]. -20 înseamnă un proces maxim prioritar, în timp ce 19 înseamnă un proces minim prioritar.

Prioritatea unui proces într-un sistem Linux se află folosind **ps** și opțiunea `nice`:

```
razvan@asgard:~$ ps -e -o pid,tty,comm,nice --sort nice
  PID TT       COMMAND          NI
    3 ?        events/0         -5
    4 ?        khelper          -5
    5 ?        kthread          -5
    8 ?        kblockd/0        -5
    9 ?        kacpid           -5
  [...]
    1 ?        init              0
  [...]
 3799 pts/4    bash              0
 4009 pts/1    ps                0
    2 ?        ksoftirqd/0      19
```

Majoritatea proceselor pornesc cu prioritatea 0. Pentru a modifica prioritatea de start a unui proces se folosește comanda **nice**, folosind argumentul `-n`:

```
razvan@asgard:~$ nice -n -10 ps -e -o pid,tty,comm,ni
nice: cannot set niceness: Permission denied

razvan@asgard:~$ nice -n 15 ps -C ps -o pid,tty,comm,ni
  PID TT       COMMAND          NI
 4051 pts/1    ps               15

asgard:/home/razvan# nice -n -10 ps -C ps -o pid,tty,comm,ni
  PID TT       COMMAND          NI
 4048 pts/2    ps              -10
```

Se poate observa că un utilizator neprivilegiat nu poate stabili o prioritate negativă (puternică), dar poate stabili una pozitivă (mai slabă). Utilizatorul `root` poate modifica prioritarea unui proces în orice direcție.

Schimbarea în timp real a priorității unui proces se poate realizeaza prin intermediul **top**. În ecranul **top** se folosește tasta `r` (*renice*). Zona de comandă oferă promptul `PID to renice:` în care se introduce pid-ul procesului a cărui prioritate va fi schimbată. După aceasta se va cere introducerea unei valori nice asociată procesului. Un utilizator neprivilegiat poate doar incrementa valoarea nice a unui proces (îl va face mai puțin prioritar).

Schimbarea priorității unui proces în Windows

În Windows, utilizatorul poate stabili prioritatea unui proces la niște valori cu granularitate mai mare: High, Very High, Normal etc. Pentru aceasta se folosește tot interfața pusă la dispoziție de `Task Manager`. Pentru alterarea prioritații unui proces, se realizează

click dreapta pe intrarea asociată procesului, se alege opțiunea `Set priority` și apoi se stabilește prioritatea dorită. Un exemplu se poate găsi în figura de mai jos:

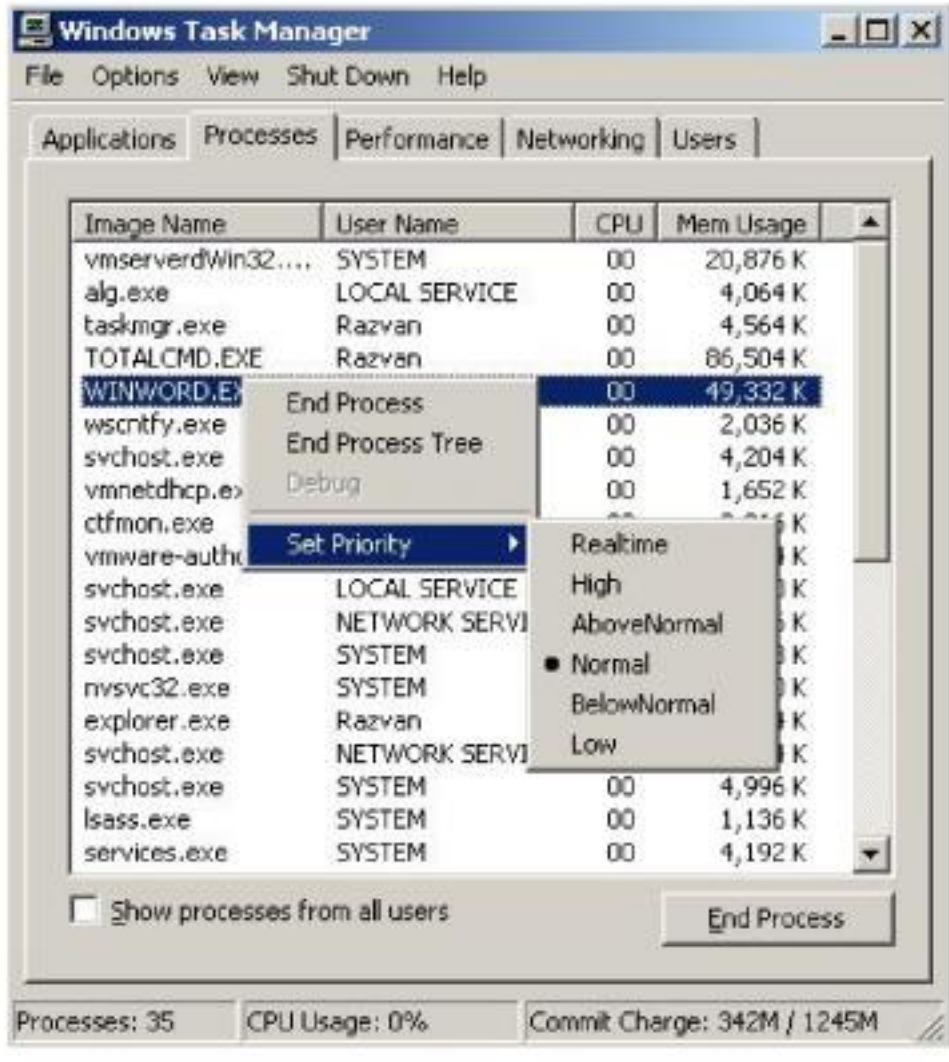


Figura 5.7: Stabilirea priorității unui proces în Windows

Cuvinte cheie

- proces
- program
- pid
- procfs
- interactivitatea unui proces
- context switch
- cuantă de timp
- multitasking
- ierarhie de procese
- init
- ps
- pstree
- pgrep
- top
- background
- foreground
- operatorul &
- bg, fg
- suspendarea unui proces
- terminal
- proces daemon
- nohup
- semnal
- kill
- killall, pkill
- CTRL-Z, CTRL-C, CTRL- CTRL-X

- operatorul |
- Task manager
- Services
- init, kswapd, pdflush, getty
- lsass, svchost, csrss
- prioritatea unui proces
- nice

Întrebări

1. Care din următoarele este un proces important Unix?
   - ❑ **ps**
   - ❑ SIGQUIT
   - ❑ **init**
   - ❑ **lsass**
2. Care utilitar NU poate fi folosit pentru a transmite semnale către un proces?
   - ❑ **kill**
   - ❑ **top**
   - ❑ **nohup**
   - ❑ **killall**
3. Câte procese **init** pot exista la un moment dat într-un sistem Linux?
   - ❑ 1
   - ❑ câte unul per procesor
   - ❑ niciunul
   - ❑ oricâte
4. Câte procese bash pot exista la un moment dat într-un sistem Linux?
   - ❑ 1
   - ❑ câte unul per procesor
   - ❑ niciunul
   - ❑ oricâte
5. Care din următoarele comenzi NU afișează PID-ul unui proces?
   - ❑ **ps**
   - ❑ **pgrep**
   - ❑ **kill**
   - ❑ **top**
6. În Linux orice program poate fi imaginea unui singur proces. Comanda top NU afișează PID-urilor proceselor din sistem.

- ❑ adevărat, adevărat
- ❑ adevărat, fals
- ❑ fals, adevărat
- ❑ fals, fals

7. Care comandă poate produce aceleași efecte ca apăsarea combinației de taste `CTRL-Z`?
   - ❑ **`pkill`**
   - ❑ **`ps`**
   - ❑ operatorul `&`
   - ❑ **`fg`**
8. Ce director din sistemul de fișiere oferă informații despre procese?
   - ❑ `/sys`
   - ❑ `/proc`
   - ❑ `/home`
   - ❑ `/`
9. În Linux procesele NU au pid-uri negative. În Linux procesele NU au priorități negative.
   - ❑ adevărat, adevărat
   - ❑ adevărat, fals
   - ❑ fals, adevărat
   - ❑ fals, fals
10. Care asociere NU este validă?
    - ❑ Task Manager – top
    - ❑ serviciu – daemon
    - ❑ explorer – nice
    - ❑ lsass – getty

Capitolul 6

Pornirea și inițializarea sistemului

Press Ctrl-Alt-Del now for IQ test.

Ce se învață din acest capitol?

- Pornirea sistemului; conceptul de bootstrapping
- Noțiunea de multitasking
- Fazele pornirii sistemului
- BIOS (POST, CMOS)
- Dispozitive boot-abile, sector bootabil (MBR/VBR)
- Rolul unui bootloader
- GRUB: rulare și configurare
- Încărcarea nucleului Linux
- Inițializarea unui sistem Linux: init/upstart și pornirea serviciilor
- Pornirea și inițializarea Windows
- Noțiuni de interoperabilitate Linux-Windows

6.1 Pornirea sistemului

Pornirea și inițializarea sistemului se referă la mecanismele prin care sistemul de operare și aplicațiile de bază ajung să fie încărcate și folosite de utilizator.

Pornirea sistemului se realizează, de obicei, prin apăsarea unui buton specific de pe unitatea centrală. În sens mai larg, ne vom referi la pornirea sistemului ca fiind o acțiune compusă ce cuprinde atât inițializarea hardware cât și cea software.

Inițializarea hardware presupune verificarea și configurarea componentelor hardware. Aceasta se execută în două situații: atunci când sistemul trece din starea oprit în starea pornit (*power on*) și atunci când sistemul este reinițializat (*reset*, *reboot*).

Acțiunea de *power on* se referă la activarea sistemului din momentul în care acesta era oprit, de cele mai multe ori prin intermediul butonului de pornire[1]. Repornirea sistemului se referă la reactivarea sistemului: din starea pornit, eventual rulând o instanță a unui sistem de operare, sistemul este repornit (fie prin intermediul butonului *Reset*, fie printr-o comandă dată sistemului de operare).

Inițializarea software presupune încărcarea sistemului de operare. Nu vom considera rularea anumitor aplicații peste sistemul de operare ca făcând parte din inițializarea software.

Pornirea sistemului prezintă un set de etape care vor fi precizate în continuare.

6.1.1 Problematica pornirii sistemului – bootstrapping

> Pornirea sistemului fizic și încărcarea sistemului de operare poartă numele de **booting** sau de **bootstrapping**.

Denumirea de *bootstrapping* își are originea în Baronul de Münchhausen. Într-o poveste a acestuia, el reușește să se salveze de la înec trăgându-se de baierele cizmelor (*boot* = cizmă, *strap* = baieră). Analogia cu această poveste se reflectă în faptul că sistemul de operare trebuie încărcat fără existența unui alt sistem de operare.

Un procesor poate executa numai cod aflat în memoria ROM[2] sau în memoria RAM. Atunci când o aplicație trebuie executată, sistemul de operare muta codul acesteia în memoria RAM și procesorul începe execuția aplicației.

Apare astfel întrebarea: cum se încarcă în memorie sistemul de operare, dacă nu există un alt sistem de operare care să îl încarce ? Aparent, trebuie "să ne tragem de baierele cizmelor" pentru "a ne scoate din apă". Vom vedea, însă, că există un set de etape care se vor parcurge succesiv pentru încărcarea diverselor componente până la încărcarea completă a sistemului de operare.

6.1.2 Etapele pornirii sistemului

Pornirea sistemului presupune încărcarea în memorie a diverselor componente până la încărcarea completă a sistemului de operare. Așa cum se poate vedea și în figura 6.1, etapele importante ale pornirii sistemului sunt:

- **încărcarea BIOS-ului**: presupune încărcarea unui program de mici dimensiuni, aflat într-o memorie dedicată de pe placa de bază, și care este utilizat pentru inițializarea componentelor hardware;

[1] Alternativ față de apăsarea butonului de pornire, un sistem poate fi pornit și de la distanță folosind WOL – http://en.wikipedia.org/wiki/Wake-on-LAN

[2] http://en.wikipedia.org/wiki/Read-only_memory

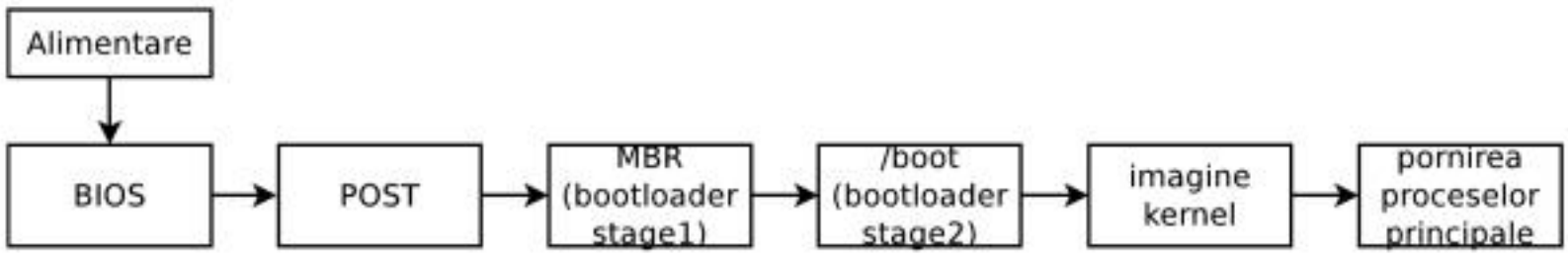


Figura 6.1: Etapele pornirii sistemului

- **rularea POST**: presupune realizarea unui set de teste hardware ce verifică funcționalitatea de bază a componentelor principale;
- **încărcarea bootloader-ului**: presupune încărcarea unui program de mici dimensiuni destinat copierii codului kernelului într-o zonă din RAM de unde să poată fi executat de procesor;
- **încărcarea nuclelui**: presupune un set de inițializări de bază pe care le realizează nucleul (kernel-ul) sistemului de operare;
- **inițializarea sistemului de operare**: presupune pornirea principalelor procese care asigura utilizarea sistemului de operare.

Etapele prezentate sunt ordonate "cronologic": o etapă este rulată și, la finalul ei, este responsabilă pentru rularea etapei ulterioare. Vor fi descrise în cadrul acestui capitol fiecare din fazele prezentate și interacțiunea între acestea.

6.1.3 BIOS

BIOS-ul (*Basic Input Output System*) este primul program încărcat în momentul pornirii sistemului. BIOS-ul este responsabil cu verificarea și inițializarea componentelor hardware.

BIOS-ul este un exemplu de **firmware**. Firmware-ul este o componentă software distribuită sub formă binară pe un suport de memorie nevolatilă. În cazul de față, BIOS-ul este stocat pe un chip de memorie ROM de pe placa de bază[1]. La *power on* sau la *reboot* BIOS-ul este rulat automat din chip-ul de pe placa de bază. BIOS-ul cunoaște specificul hardware al sistemului și devine responsabil cu inițializarea componentelor fizice ale acestuia.

CMOS

Deși este un program mic și limitat la specificul hardware al sistemului, BIOS-ul oferă optiuni de configurare. Configurările BIOS-ului sunt păstrate într-un chip separat de memorie RAM nevolatilă alimentată prin intermediul unei baterii. Aceast chip de memorie se numește CMOS[2] – *Complementary Metal-Oxide Semiconductor*. Atât chip-ul cât și bateria se găsesc pe placa de bază. În momentul în care bateria este scoasă, configurările BIOS-ului se pierd revenindu-se la configurările implicite.

[1] Memoria este de tip EPROM – http://en.wikipedia.org/wiki/EPROM
[2] http://en.wikipedia.org/wiki/Cmos

Figura 6.2: Ecran de configurare CMOS

Configurarea BIOS-ului se realizează prin intermediul unui ecran de configurare. Opțiunile disponibile includ stabilirea unei parole pe BIOS, selectarea dispozitivului de boot (vezi secțiunea 6.2.1), alterarea frecvenței procesorului și alte configurări care depind de placa de bază și de componentele hardware existente în sistem.

6.1.4 POST

Etapa de verificare și de inițializarea componentelor hardware ale sistemului (procesor, memorie, placă video) poartă numele de POST (*Power-on Self Test*). Dacă una dintre componente prezintă erori și nu poate fi inițializată corespunzător, atunci BIOS-ul va emite un set de sunete specifice și nu se va continua inițializarea sistemului.

După verificarea și inițializarea componentelor, BIOS-ul este responsabil cu încărcarea bootloader-ului.

6.2 Bootloader

Bootloader-ul este componenta utilizată în principal pentru încărcarea imaginii de kernel într-o zonă din memori RAM, de unde să poată fi executată de către procesor. Cele mai multe *bootloader*-e permit utilizatorului să aleagă între mai multe versiuni de kernel ce vor fi încărcate, selectând astfel între mai multe sisteme de operare. De asemenea, *bootloader*-ul permite și transmiterea unor opțiuni de încărcare a nucleului sistemului de operare. *Bootloader*-ul este încărcat în memorie de BIOS.

Bootloader-ul se regăsește pe primul sector al unui dispozitiv bootabil, denumit și **sectorul de boot**. Aceasta limitare de dimensiune apare deoarece BIOS-ul poate încărca în memorie un singur sector (512 octeți). Se poate întâmpla ca *bootloader*-ul să fie prea mare și să nu încapă în primul sector al dispozitivului boot-abil. În acest caz

Figura 6.3: BIOS POST (Power-On Self Test)

bootloader-ul se împarte în două componente. Prima componentă începe în sectorul de boot. A doua, *second stage*, rezidentă pe un spațiu mai mare de pe hard-disk și este încărcată de către prima. Un astfel de multiple-stage bootloader este GRUB, bootloader-ul implicit pe majoritatea distribuțiilor Linux.

6.2.1 Dispozitive boot-abile

Un dispozitiv boot-abil este un dispozitiv al cărui prim sector este un **sector boot-abil**. Sectorul boot-abil se caracterizează prin faptul că ultimii doi octeți sunt `0xAA55` (din cei 512).

Exemple de dispozitive ce pot fi boot-abile sunt CD-ROM-uri, hard-disk-uri, floppy disk-uri, USB flash etc. În cazul acestor dispozitive, dacă primul sector are structura specifică unui sector boot-abil atunci poate fi folosit pentru încărcarea altor informații existente.

După rularea POST, BIOS-ul consultă CMOS și urmează ordinea de boot-are de acolo. Astfel, dacă CMOS-ul a fost configurat ca în imaginea de mai sus, BIOS-ul va efectua următoarele operații:

- dacă HDD-ul are un prim sector bootabil, se va încărca și se va executa conținutul sectorului; aici se va afla, de obicei, bootloader-ul; în urma execuției se va încărca sistemul de operare;
- dacă HDD-ul nu are un prin sector boot-abil, se trece la investigația CD-ROM-ului din sistem;
- se va verifica existența unui CD în unitatea de CD-ROM; dacă acesta există, se va verifica existența sectorului de boot;

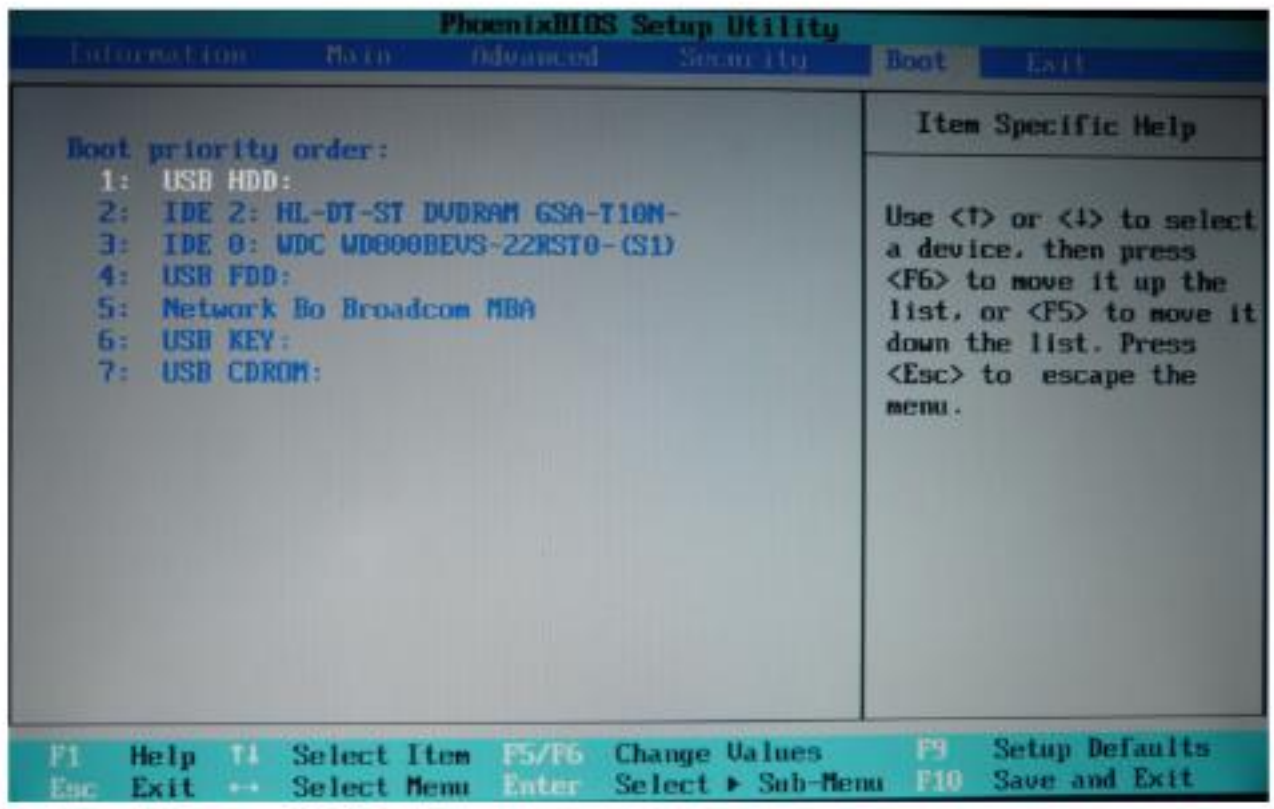


Figura 6.4: Alegerea dispozitivului de boot

- dacă CD-ul are un prim sector bootabil, atunci se va încărca și se va executa conținutul sectorului de boot; exemple de CD boot-abil sunt CD-ul de instalare a unui sistem de operare sau un LiveCD;
- dacă nici CD-ROM-ul nu are un prim sector bootabil atunci se trece la următorul dispozitiv boot-abil (al doilea HDD), care va fi analizat asemenea celor aterioare;
- dacă nu există niciun dispozitiv boot-abil, nu se poate încărca niciun sistem de operare; BIOS-ul va afișa un mesaj specific fără a executa nimic.

6.2.2 Structura sectorului de boot pentru un dispozitiv boot-abil

Primul sector (primii 512 octeți) al unui dispozitiv boot-abil poartă denumirea de **MBR** (*Master Boot Record*). Primul sector al unei partiții al unui dispozitiv boot-abil partiționat poartă numele de **VBR** (*Volume Boot Record*). După cum a fost specificat anterior, un dispozitiv este boot-abil dacă MBR-ul acestuia este un sector boot-abil (are ultimii doi octeți `0xAA55`, pentru a semnifica faptul că sectorul conține secvențe de cod executabile).

Sectorul de boot poate conține instrucțiuni pentru încărcarea unei secvențe mai mari de cod (cum este sectorul de boot al unui CD), sau poate conține un bootloader pentru încărcarea sistemului de operare (cum este sectorul de boot al unui hard-disk).

Conținutul sectorului de boot este încărcat în memorie de BIOS. După aceasta, BIOS-ul transmite controlul programului rezident în sectorul de boot (*bootloader*).

În figura 6.5 este prezentată structura MBR.

După cum se observă, există următoarele zone importante:

- **zona de cod executabil**; aceasta conține programul căruia îi este transmis controlul de către BIOS (*bootloader*);

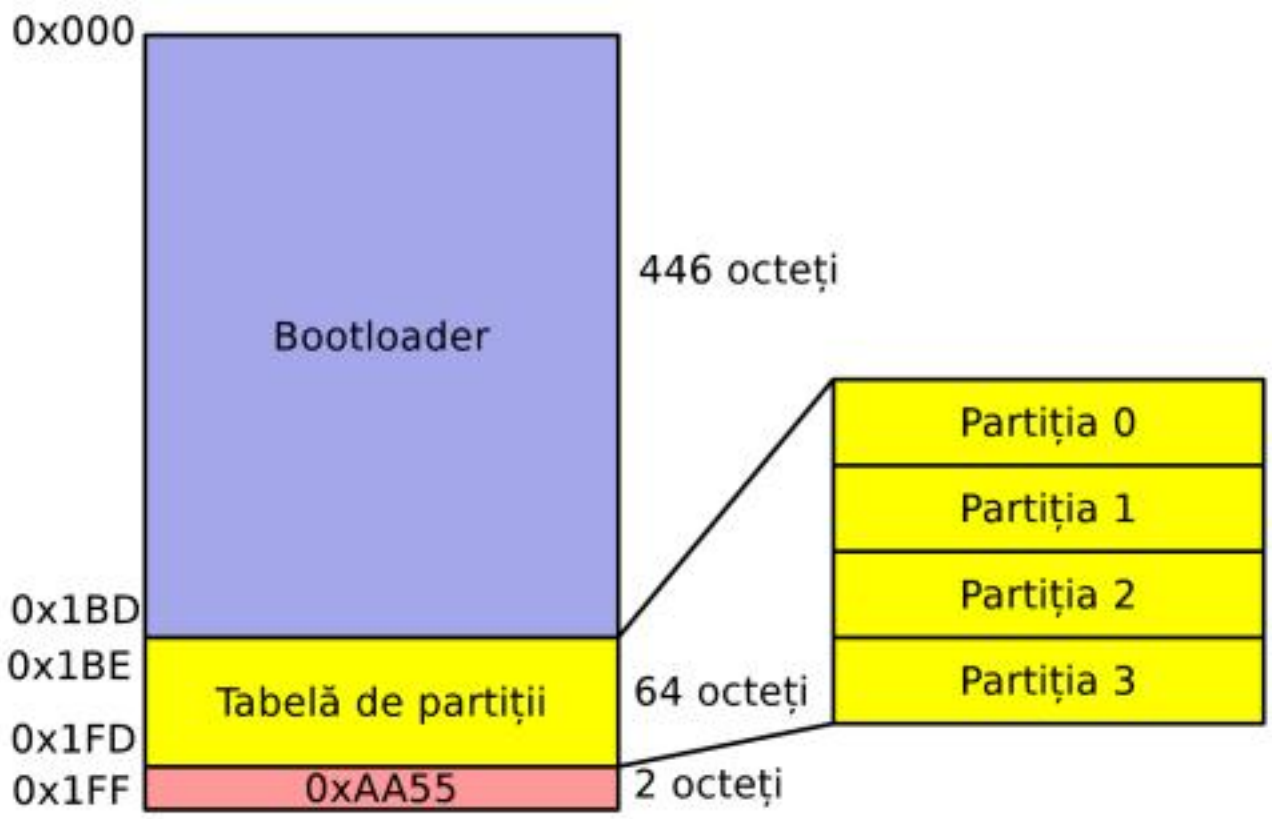


Figura 6.5: Structura MBR (Master Boot Record)

- **tabela de partiții** conține informații despre partițiile din sistem; fiecărei partiții îi corespunde o intrare de 16 octeți;
- **semnătura** de sector boot-abil (`0xAA55`).

Zona de cod executabil conține programul care va fi rulat după încărcarea MBR-ului în memorie de BIOS. De obicei, acest program este *bootloader*-ul, responsabil cu descoperirea partiției active și încărcarea primului sector (VBR) de pe aceasta. În cazul existenței mai multor sisteme de operare, *bootloader*-ul poate permite utilizatorului să opteze pentru încărcarea unui sistem de operare aflat pe altă partiție decât cea de boot.

Tabela de partiții

Tabela de partiții rezidă în MBR și ocupă 64 de octeți. Un hard-disk poate conține doar patru partiții (așa numitele **partiții primare**), rezultând 16 octeți pentru descrierea fiecărei partiții.

Cei 16 octeți ai fiecărei partiții oferă, pe lângă adresa de start (exprimată în *cylinder-head-sector*[1]) și dimensiunea și tipul partiției. O partiție poate fi activă sau non-activă. O partiție activă este aleasă de *bootloader*-ul aflat în zona MBR (sau de către o parte a *bootloader*-ului, dacă acesta este în două stagii) pentru a continua încărcarea sistemului. De pe o partiție activă se va încărca primul sector (VBR) al acesteia.

O altă clasificare a partițiilor este clasificarea în partiții primare și partiții extinse. Numărul de 4 **partiții primare** care pot fi descrise în tabela de partiții din MBR este, de multe ori, insuficient. Pentru a rezolva acest neajuns, a apărut noțiunea de **partiție extinsă**. Pe un sistem poate exista o singură partiție extinsă astfel că putem avea fie maxim 4 partiții primare, fie o partiție extinsă și maxim 3 primare.

[1] http://en.wikipedia.org/wiki/Cylinder-head-sector

Partițiile extinse pot conține mai multe **partiții logice**, mărind astfel numărul de partiții posibile din sistem. Descrierea partiției extinse se regăsește în primul sector din acea partiție, denumit **EPBR** (*Extended Partition Boot Record*).

Partițiile sunt folosite pentru separația informațiilor într-un sistem de operare. Pentru a asigura existența unui sistem de fișiere pe o partiție, acea partiție va trebui formatată (vezi secțiunea 4.8.1).

Salvarea sectorului de boot (backup)

Uneori se dorește salvarea sectorului de boot al unui disc sau al unei partiții pentru situația în care acesta devine corupt și trebuie reparat. În Linux, acest lucru se poate realiza foarte ușor prin intermediul comenzii **dd** (vezi secțiunea 7.6.5) și a dispozitivelor de tip bloc (vezi secțiunea 7.6.1)

Astfel, dacă se dorește realizarea unei copii locale a MBR, se poate folosi utilitarul **dd** și dispozitivul asociat hard-disk-ului (în cazul nostru `/dev/hda`). Vor trebui copiați 512 octeți (un sector), deci contorul de sectoare va fi 1:

```
1 root@asgard:~# dd if=/dev/hda of=mbr.out count=1
2 1+0 records in
3 1+0 records out
4 512 bytes (512 B) copied, 0.035273 seconds, 14.5 kB/s
5
6 root@asgard:~# ls -l mbr.out
7 -rw-r--r-- 1 root root 512 2009-08-07 23:28 mbr.out
```

6.2.3 Mecanismul de funcționare a unui bootloader

După cum s-a specificat, un *bootloader* este un program folosit pentru a încărca în RAM imaginea sistemului de operare, sau alte componente care vor încărca (la rândul lor) acea imagine.

Codul *bootloader*-ului este supus unor constrângeri destul de rigide, întrucât trebuie să încapă în MBR. Mai exact, este limitat la 446 de octeți (restul fiind ocupați de tabela de partiții și de semnătura de boot `0xAA55`).

Ca atare *bootloader*-ele lucrează în mod normal în două etape (*stages*). Prima etapă este cea care rezidă în MBR. Aceasta are rolul de a încărca de pe hard-disk a doua etapă care este suficient de complexă pentru a putea începe încărcarea efectivă a kernelului sistemului de operare. În cea de-a doua etapă poate fi oferit un ecran de selecție pentru situația în care coexistă două sisteme de operare pe același sistem fizic. Unele *bootloader*-e (spre exemplu GRUB versiunea 1) au o fază intermediară – 1.5 – pentru a permitea citirea de informații de pe hard-disk dincolo de o anumită limită.

După acest pas bootloader-ul va încărca nucleului sistemului de operare (sau o parte a acestuia) și va transmite controlul către acesta.

Exemple de bootloader-e sunt:

- **GRUB** (*GRand Unified Bootloader*): *bootloader*-ul implicit pe majoritatea distribuțiilor Linux;

- **LILO** (*Linux Loader*): *bootloader*-ul implicit pe distribuțiile Linux înainte de supremația GRUB;
- **NTLDR** (*NT Loader*): *bootloader*-ul folosit de Windows NT, 2000, XP, 2003;
- **winload.exe**: *bootloader*-ul folosit de Windows Vista;
- **SYSLINUX**: o suită de *bootloader*-e destinate diverselor medii de boot.

Chainloading

Modul cel mai des întâlnit prin care un sistem de operare este inițializat este prin specificarea directă a imaginii nucleului către *bootloader*.

Alternativa o reprezintă boot-area indirectă a sistemului de operare. Acest lucru presupune ca *bootloader*-ul aflat în MBR, în loc de imaginea unui kernel, să încarce un alt *bootloader* care va încărca la rândul său imaginea de kernel a sistemului de operare. Acest mod de încarcare a unui sistem de operare se numește **chainloading**.

Bootloader-ele specifice Linux (precum GRUB sau LILO) încarcă sistemele Windows folosind *chainloading*. Spre exemplu, dacă după încărcarea *bootloader*-ului GRUB se optează pentru boot-area Windows, atunci se va incărca *bootloader*-ul Windows (NTLDR) care va inițializa la rândul său sistemul de operare.

6.2.4 GRUB

GRUB (*GRand Unified Booloader*[1]) este bootloader-ul implicit pe distribuțiile Linux și alte sisteme de operare din familia Unix și este, de obicei, folosit în situațiile de **multiboot** (când pe sistemul de calcul coexistă mai multe sisteme de operare). Această secțiune va prezenta detalii despre versiunea 1 a GRUB, denumită simplu GRUB. Versiunea 2, denumită GRUB 2, care a început să fie utilizată recent, este prezentată în cadrul unui studiu de caz (vezi secțiunea 6.6.1).

Specific GRUB este faptul că există o fază intermediară 1.5 pentru situația în care există o limită de vizibilitate a hard-disk-ului.

Ca și în cazul altor *bootloader*-e, prima fază (cea care rezidă în MBR) are rolul de a încărca faza a doua sau, dacă este nevoie, faza 1.5. Datorită dimensiunii reduse, prima fază nu deține informații despre sistemul de fișiere. în schimb, faza 1.5 deține informații despre acesta și poate localiza și încărca în mod corespunzător faza a doua.

Faza a doua este responsabilă cu citirea fișierului principal de configurare (`/boot/grub/menu.lst`), afișarea ecranului de opțiuni și încărcarea nucleului sistemului de operare.

În figura 6.6 este prezentată localizarea diferitelor faze ale GRUB pe hard-disk.

Se observă că prima etapă (*grub stage 1*) se găsește în zona de cod executabil din MBR. Faza 1.5 (*grub stage 1.5*) se găsește în zona de compatibilitate a hard-disk-ului (primii 32 KB). Această etapă recunoaște tipul sistemului de fișiere pe care se află etapa

[1] http://www.gnu.org/software/grub/

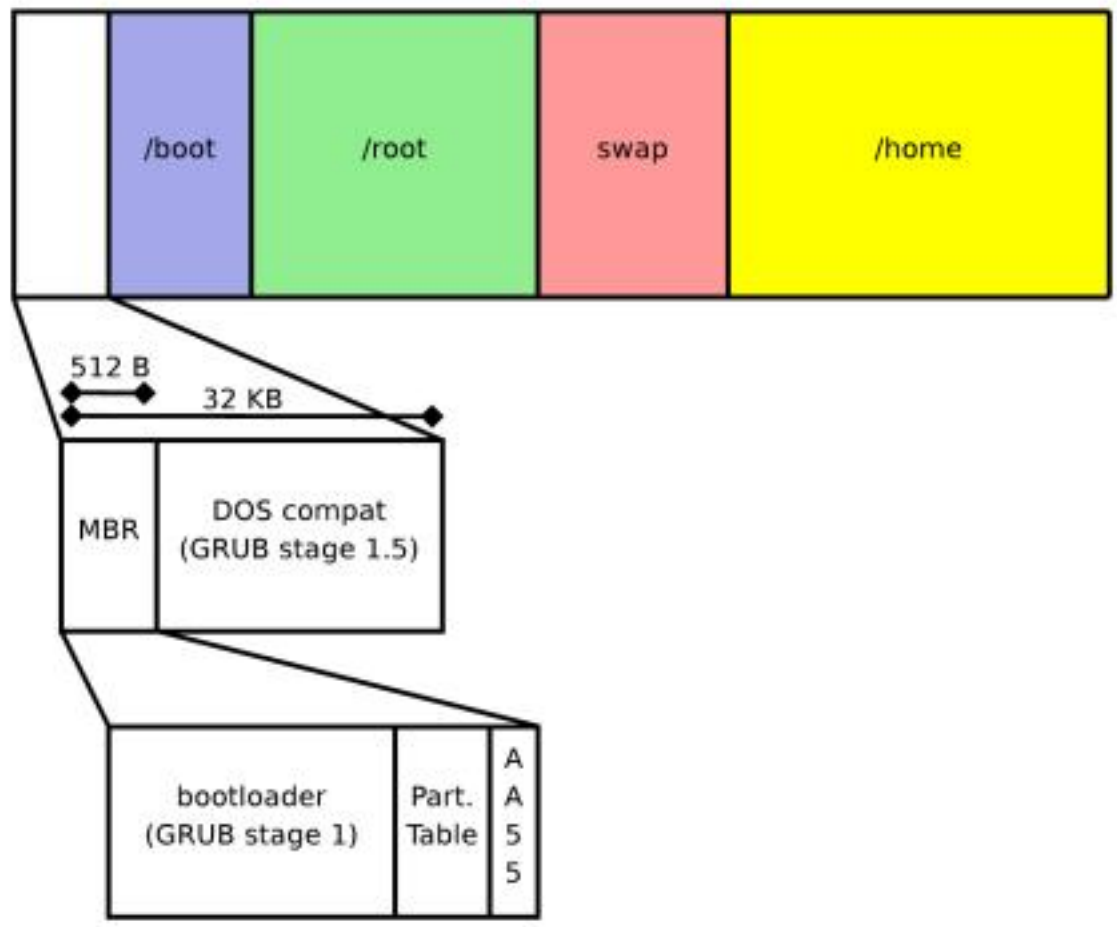


Figura 6.6: Structura GRUB

a doua și este utilizată pentru încărcarea acesteia. Etapa a doua rezidă în partiția de boot.

Fișierele asociate GRUB se găsesc pe un sistem Debian/Ubuntu, în `/boot/grub`, așa cum este prezentat mai jos:

```
1  razvan@asgard:/boot/grub$ ls
2  default         fat_stage1_5  minix_stage1_5     stage2
3  device.map      jfs_stage1_5  reiserfs_stage1_5  xfs_stage1_5
4  e2fs_stage1_5   menu.lst      stage1
5
6  razvan@asgard:/boot/grub$ ls -l
7  total 188
8  -rw-r--r-- 1 root root    197 2009-01-17 11:55 default
9  -rw-r--r-- 1 root root     15 2009-01-17 11:55 device.map
10 -rw-r--r-- 1 root root   7584 2009-01-17 11:55 e2fs_stage1_5
11 -rw-r--r-- 1 root root   7424 2009-01-17 11:55 fat_stage1_5
12 -rw-r--r-- 1 root root   8192 2009-01-17 11:55 jfs_stage1_5
13 -rw-r--r-- 1 root root   4724 2009-06-23 12:45 menu.lst
14 -rw-r--r-- 1 root root   6848 2009-01-17 11:55 minix_stage1_5
15 -rw-r--r-- 1 root root   9280 2009-01-17 11:55 reiserfs_stage1_5
16 -rw-r--r-- 1 root root    512 2009-01-17 11:55 stage1
17 -rw-r--r-- 1 root root 108392 2009-01-17 11:55 stage2
18 -rw-r--r-- 1 root root   8904 2009-01-17 11:55 xfs_stage1_5
```

Se observă că prima fază se găsește în fișierul `stage1` care are exact dimensiunea unui sector (512 octeți). Faza 1.5 are o formă specializată pentru fiecare sistem de fișiere (`e2fs_stage1_5`, `reiserfs_stage1_5` etc.). Spațiul ocupat este în jur de 7-8 KB, deci încape fără probleme în zona de compatibilitate de 32 KB a hard-disk-ului.

Faza a doua este descrisă de fișierul `stage2` cu dimensiune mai mare decât a celorlalte (circa 100 KB). Faza a doua este resposabilă pentru citirea fișierului de configurare `menu.lst` și încărcarea ulterioară a nucleului.

Vizualizare stage1 și stage1.5 pe hard-disk

Pentru a verifica prezența primei faze și a fazei 1.5 pe hard-disk vom folosi utilitarele **dd** și **hexdump**.

Într-o primă fază va fi realizată o copie a primului sector de pe hard-disk în fișierul `first_sector` și apoi a următoarelor 10 sectoare în fișierul `next_sectors`.

```
root@asgard:/boot/grub# dd if=/dev/hda of=first_sector count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 8.91e-05 seconds, 5.7 MB/s

root@asgard:/boot/grub# dd if=/dev/hda of=next_sectors count=10 skip=1
10+0 records in
10+0 records out
5120 bytes (5.1 kB) copied, 0.000137291 seconds, 37.3 MB/s
```

În `first_sector` și `next_sector` se găsesc informațiile binare conținute în primele sectoare ale hard-disk-ului. Ele vor fi comparate cu fișierele imagine din `/boot/grub` folosind utilitarul **hexdump**. De exemplu, pentru `stage1`:

```
razvan@asgard:/boot/grub$ hexdump first_sector
[...]
0000070 34e8 f601 80c2 5474 41b4 aabb cd55 5a13
0000080 7252 8149 55fb 75aa a043 7c41 c084 0575
0000090 e183 7401 6637 4c8b be10 7c05 44c6 01ff
00000a0 8b66 441e c77c 1004 c700 0244 0001 8966
[...]
0000170 be06 7d94 30e8 be00 7d99 2ae8 eb00 47fe
0000180 5552 2042 4700 6f65 006d 6148 6472 4420
0000190 7369 006b 6552 6461 2000 7245 6f72 0072
00001a0 01bb b400 cd0e ac10 003c f475 00c3 0000
[...]

razvan@asgard:/boot/grub$ hexdump stage1
[...]
0000070 34e8 f601 80c2 5474 41b4 aabb cd55 5a13
0000080 7252 8149 55fb 75aa a043 7c41 c084 0575
0000090 e183 7401 6637 4c8b be10 7c05 44c6 01ff
00000a0 8b66 441e c77c 1004 c700 0244 0001 8966
[...]
0000170 be06 7d94 30e8 be00 7d99 2ae8 eb00 47fe
0000180 5552 2042 4700 6f65 006d 6148 6472 4420
0000190 7369 006b 6552 6461 2000 7245 6f72 0072
00001a0 01bb b400 cd0e ac10 003c f475 00c3 0000
[...]
```

După cum se observă, în primul sector al hard-disk-ului se găsește imaginea primei etape a GRUB, iar în următoarele sectoare imaginea etapei 1.5. Dacă apar deosebiri între imaginea aflată în sectoarele discului și cea din `/boot/grub`, acestea sunt din considerente de adaptare la geometria discului. Imaginea celei de-a doua faze este încărcată de faza 1.5 din `/boot/grub/stage2`, aceasta putând accesa structura sistemului de fișiere.

Creare unei dischete de boot GRUB

Deși dischetele sunt dispozitive pe cale de dispariție (și costă în ziua de azi mai mult decât un CD), vom descrie în scop didactic modul în care se poate crea o dischetă de boot GRUB.

Discheta va trebui să conțină imaginea pentru prima și cea de-a doua fază (nu este nevoie de faza 1.5 întrucât nu avem limitare în cest caz). Pentru aceasta se poate folosi folosi utilitarul **cat**:

```
root@anaconda:/boot/grub# cat stage1 stage2 > /dev/fd0
```

sau utilitarul **dd**:

```
root@anaconda:/boot/grub# dd if=stage1 of=/dev/fd0
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.0426716 seconds, 12.0 kB/s

root@anaconda:/boot/grub# dd if=stage2 of=/dev/fd0 skip=1
210+1 records in
210+1 records out
107656 bytes (108 kB) copied, 0.020834 seconds, 5.2 MB/s
```

Dacă se configurează BIOS-ul pentru a boot-a de pe dischetă, utilizatorului îi va fi oferită consola GRUB de configurare (vezi secțiunea 6.3).

6.3 Configurarea GRUB

Configurarea GRUB se realizează prin intermediul fișierului `/boot/grub/menu.lst`. În acest fișier sunt precizate opțiunile de boot prezente în meniul GRUB, cu alte cuvinte sistemele de operare care pot fi încărcate cu ajutorul GRUB. Etapa a doua citește acest fișier de configurare și oferă utilizatorului ecranul de opțiuni:

În continuare este prezentată o parte dintr-un fișier de configurare `/boot/grub/menu.lst` și sunt descrise elementele de configurare prezente în acesta:

```
## default num
# Set the default entry to the entry number NUM. Numbering starts from 0,
 and
# the entry number 0 is the default if the command is not used.
[...]
default         0

## timeout sec
# Set a timeout, in SEC seconds, before automatically booting the default
 entry
# (normally the first entry defined).

timeout         5

# Pretty colours
color cyan/blue white/blue
[...]
```

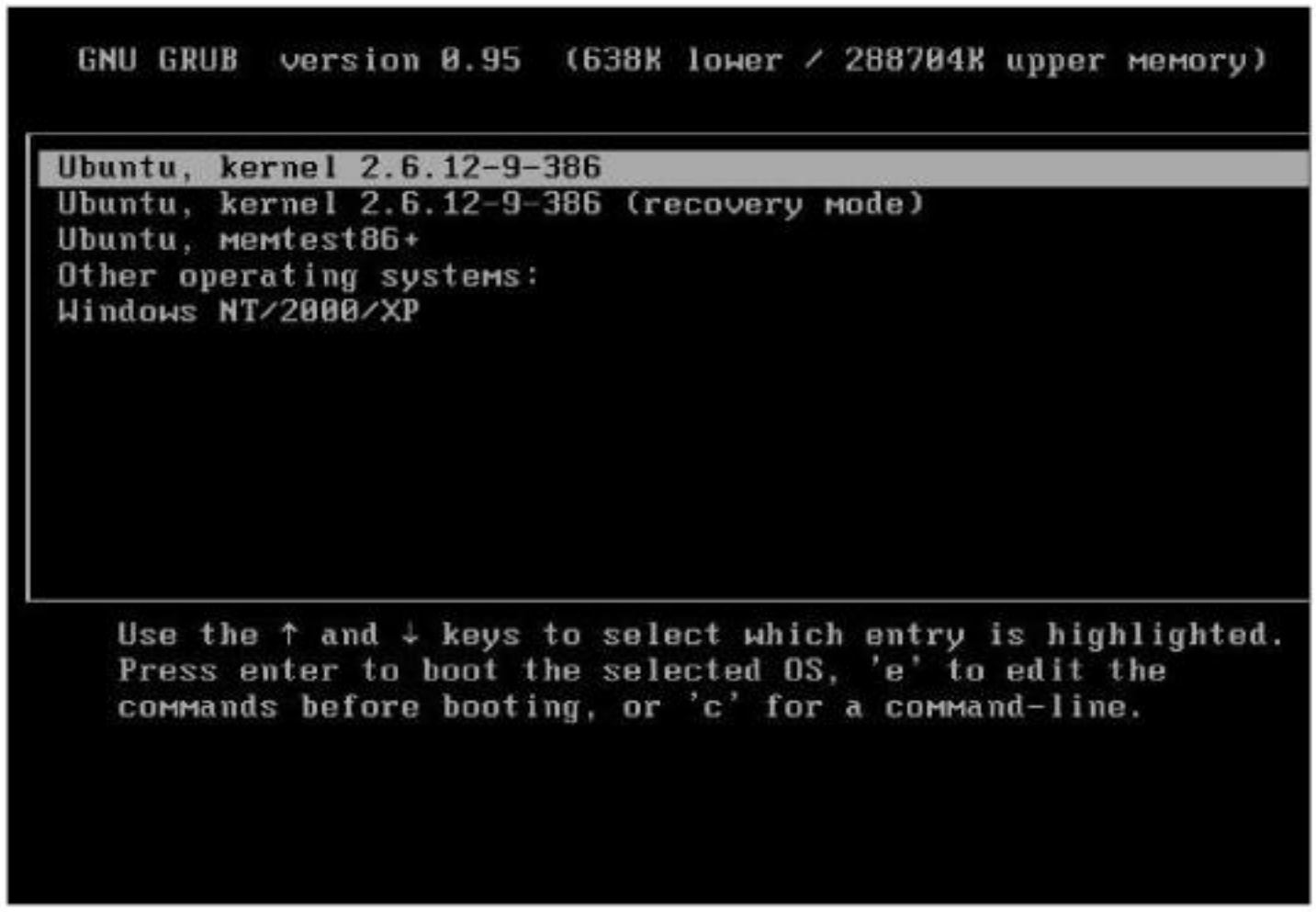


Figura 6.7: Ecran opțiuni GRUB

```
16
17 ## Razvan's kernels
18
19 title           Debian GNU/Linux, kernel 2.6.18-12fast
20 root            (hd0,2)
21 kernel          /boot/vmlinuz-2.6.18-12fast root=/dev/hda3 ro
22 savedefault
23
24 title           Debian GNU/Linux, kernel 2.6.18-12fast (single-user mode)
25 root            (hd0,2)
26 kernel          /boot/vmlinuz-2.6.16-12fast root=/dev/hda3 ro single
27 savedefault
28 [...]
29
30 # This entry automatically added by the Debian installer for a non-linux
   OS
31 # on /dev/hda1
32 title           Microsoft Windows XP Professional
33 root            (hd0,0)
34 savedefault
35 makeactive
36 chainloader     +1
```

Opțiunea `default` precizează opțiunea implicită din meniu folosită pentru boot-are. Fișierul de configurare conține mai multe intrări pentru diferite sisteme de operare. Numărul care urmează după opțiunea `default` indică sistemul de operare care va fi boot-at implicit în cazul în care utilizatorul nu intervine în momentul afișarii meniului de opțiuni. Numerotarea pornește de la 0. Presupunând că, în figura 6.7, nu ar exista alte opțiuni de sisteme de operare, boot-area implicită pe Windows XP Professional este condiționată de prezența opțiunii default 4.

Opțiunea `timeout` indică timpul de așteptare pentru ca utilizatorul să aleagă una din

opțiunile afișate în meniul de boot. Dacă acest timp expiră, se va încărca sistemul de operare indicat de opțiunea `default`.

Opțiunea `colors` afișează colorat meniul GRUB.

Opțiunea de sistem de operare

Ceea ce apare, în continuare, în exemplul de fișier de configurare de mai sus sunt intrările asociate pentru sistemele de operare care pot fi boot-ate. Se observă că fiecare intrare conține un set de directive de configurare. Acestea sunt:

- `title`: este un șir de caractere reprezentând numele intrării așa cum va apărea ea în ecranul de opțiuni GRUB. Deși nu există nicio limitare de nume, se recomandă ca titlul să fie un nume reprezentativ;
- `root`: reprezintă partiția pe care se află imaginea de nucleu a sistemului de operare; aceasta poate sau nu coincidă cu partiția unde este montat sistemul de fișiere rădăcină (`/`);
- `kernel`: reprezintă calea către imaginia de nucleu în cadrul partiției de mai sus; subdirectiva `root` precizează partiția asociată sistemului de fișiere rădăcină (`/`); opțiunea `ro` precizează faptul că sistemul de fișiere va fi montat read-only; se observă că în cazul Windows nu există opțiunea `kernel`, deorece GRUB nu poate boot-a direct nucleul Windows ci va apela NTLDR (*bootloader*-ul de Windows), lucru indicat prin opțiunea `chainloader+1`; opțiunea `+1` semnifică citirea primului sector din partiția `(hd0,0)`;
- opțiunea `makeactive` stabilește partiția ca fiind partiția activă;
- opțiunea `savedefault`: dacă directiva `default` este folosită în forma `default saved` și opțiunea folosită la boot-are are opțiunea `savedefault`, atunci la următoarea boot-are se va selecta, implicit, aceeași opțiune.

Convenția de denumire GRUB

Dacă în Linux, primul hard-disk este accesibil sub numele `/dev/hda`, iar prima partiție a acestuia sub numele `/dev/hda1`, GRUB folosește o altă convenție pentru denumirea discurilor din sistem și a partițiilor acestora. Astfel, pentru a referi primul disc din sistem, GRUB folosește denumirea `(hd0)`. Numerotarea discurilor începe de la 0. Spre deosebire de Linux, GRUB nu face distincție între tipul de controller folosit de discurile din sistem: IDE sau SCSI. Astfel, primul disc este denumit întotdeauna `(hd0)`. Pentru referirea primei partiții din primul disc, GRUB folosește denumirea `(hd0,0)`.

În ton cu cele de mai sus, a 3-a partiție de pe al doilea disc al sistemului este referită de GRUB sub forma `(hd1,2)`. Tabelul 6.1 de mai sus oferă o analogie între denumirea GRUB și Linux pentru discuri și partiții.

Consola GRUB

În afara fișierului de configurare, GRUB poate fi configurat și prin intermediul unei consolei asociate. Aceasta poate fi accesată în două moduri: dintr-un sistem Linux

Tabelul 6.1: Denumirea discurilor în Linux/GRUB

Dispozitiv	**Denumire Linux**	**Denumire GRUB**
Primul disc IDE	`/dev/hda`	`(hd0)`
Al doilea disc IDE	`/dev/hdb`	`(hd1)`
Al treilea disc SCSI	`/dev/sdc`	`(hd2)`
Prima unitate floppy	`/dev/fd0`	`(fd0)`
Prima partitie de pe al doilea disc IDE	`/dev/hdb1`	`(hd1,0)`
A doua partitie de pe al treilea disc SCSI	`/dev/sdc2`	`(hd2,1)`

folosind comanda **`grub`**, sau din intermediul meniului de boot al GRUB. În ambele cazuri utilizatorului îi este oferit un prompt `grub>` unde poate introduce comenzi specifice.

Utilizarea consolei GRUB este limitată la situațiile de testare a unor opțiuni, de recuperare sau de troubleshooting.

Un exemplu util de utilizare a consolei GRUB este (re)instalarea GRUB, așa cum este descrisă în secțiunea 6.6.3.

Accesarea consolei GRUB poate fi necesară și în momentul în care se dorește alterarea unei opțiuni de boot-are. Spre exemplu, în ecranul de meniul de boot GRUB, apăsarea tastei `e` peste o anumită opțiune conduce la apariția consolei GRUB cu posibilitatea editării acelei opțiuni de bootare. Acest lucru este util în cazul unei erori (s-a precizat de exemplu un nume greșit pentru imaginea de kernel) sau în cazul în care s-a pierdut parola de root și se dorește recuperarea acesteia, forțând sistemul să pornească o consolă de root (prin folosirea opțiunii `init=/bin/bash` (vezi secțiunea 10.6.2).

6.4 Încărcarea nucleului

După ce este încărcat în RAM și rulat, *bootloader*-ul este responsabil cu încărcarea nucleului sistemului de operare și transmiterea controlului către acesta. Ca și faza a doua a GRUB, nucleul rezidă într-un fișier din sistemul de fișiere, denumit **imaginea de nucleu**. În cazul GRUB, locația acestui fișier este transmisă ca parametru în cadrul directivei `kernel` din `/boot/grub/menu.lst`.

6.4.1 Imaginea de nucleu

Imaginea de nucleu este un fișier executabil conținând codul și datele necesare pentru rularea nucleului sistemului de operare. În Windows, imaginea nucleului se numește `ntoskrnl.exe` și este localizată în `C:\Windows\system32\ntoskrnl.exe`. În sistemele Linux, această imagine rezidă, de obicei, în `/boot`, și are denumirea în forma `vmlinuz-versiune_kernel` (spre exemplu, `vmlinuz-2.6.29.1` sau `vmlinuz-2.6.31-020631-generic`).

Comanda **`file`** indică un astfel de fișier ca fiind o imagine de nucleu Linux:

```
1 root@asgard:/boot# file vmlinuz-2.6.31-020631-generic
2 vmlinuz-2.6.31-020631-generic: Linux kernel x86 boot executable RO-rootFS
  , root_dev 0x6801, swap_dev 0x3, Normal VGA
```

În mod obișnuit imaginea de kernel ocupă în jur de 3-4 MB. Poate fi însă constrânsă la dimensiuni mai mici prin eliminarea anumitor componente, atunci când este cazul (spre exemplu în cazul sistemelor *embedded*[1]). Faptul că denumirea imaginii se termină în `z` (`vmlinuz`) indică o imagine de kernel comprimată.

O imagine de kernel comprimată conține la începutul ei o secțiune direct executabilă, și după această secțiune, se găsește zona efectiv comprimată.

După încărcarea imaginii de nucleu, *bootloader*-ul transmite controlul execuției către kernel (fie către imaginea necomprimată, fie către zona necomprimată). Într-o primă fază nucleul inițializează și configurează dispozitivele hardware și memoria. Toate dispozitivele inițializate de BIOS vor fi reinițializate pentru a asigura robustețea și independența de modul în care BIOS-ul realizează initializarea.

Următorul pas îl reprezintă decomprimarea imaginii de nucleu. După aceasta se montează sistemul de fișiere rădăcină precizat prin parametrul `root=/dev/...` al directivei `kernel` din fișierul de configurare GRUB. Pe măsură ce nucleul inițializează alte subsisteme, pe ecran sunt afișate mesaje de status.

Încărcarea completă a nucleului se încheie cu crearea procesului `init` (ce are pid-ul 1) din imaginea `/sbin/init`. Se poate alege o altă imagine pe baza căreia să se creeze procesul `init`, prin specificarea opțiunii `init=/path/to/new/exec` la boot-area nucleului. Un caz tipic pentru aceasta este recuperarea parolei de root (vezi secțiunea 10.6.2).

6.4.2 Opțiuni de boot-are pentru nucleu

Nucleul Linux poate fi configurat pentru boot-are cu diverse argumente care să-i modifice funcționalitatea. Acestea sunt transmise ca parametri de boot-are în directiva `kernel` a GRUB. Configurarea acestor argumente poate fi realizată fie prin fișierul de configurare (`/boot/grub/menu.lst`), fie prin accesarea consolei GRUB (apăsarea tastei `e` în ecranul de opțiuni GRUB).

O listă completă cu opțiunile de boot-are poate fi găsită online[2]. Cele mai importante dintre aceste opțiuni sunt:

- **debug**: afișează mesaje suplimentare la încărcarea nucleului;
- **quiet**: dezactivează afișarea de mesaje la încărcarea nucleului;
- **resume=suspend_device** (util pentru laptop-uri în cazul fincției de hibernare): specifică discul unde se află imaginea suspendată a nucleului;
- **root=device**: precizează partiția unde se află sistemul de fișiere rădăcină;

[1] http://en.wikipedia.org/wiki/Embedded_system
[2] http://www.kernel.org/pub/linux/kernel/people/gregkh/lkn/lkn_pdf/ch09.pdf

- **ro/rw**: specifică dacă sistemul de fișiere rădăcină este montat *read-only* sau *read-write* la boot-are;
- **init=filename**: specifică executabilul care va fi încărcat la crearea procesului `init` (implicit acest executabil este `/sbin/init`);
- **S** sau **single**: specifică pornirea lui `init` în mod *single-user*.

6.5 Inițializarea sistemului

Încărcarea nucleului se încheie cu pornirea `init`, procesul cu pid-ul 1, părintele celorlalte procese din sistem. În cele ce urmează vom presupune că procesul init a fost creat în mod obișnuit din imaginea `/sbin/init`.

`init` este responsabil cu pornirea celorlalte procese importante ale sistemului, a daemonilor și a proceselor care asigură accesul utilizatorului la sistem (`getty` și mediul grafic).

6.5.1 init

`init` este resposabil cu crearea celorlalte procese importante din sistem și, astfel, de realizarea unui sistem funcțional și interactiv. Pentru precizarea proceselor care trebuiesc pornite și a opțiunilor primite de către acestea, `init` folosește un set de fișiere de configurare decrise în continuare.

Niveluri de rulare (runlevels)

Înainte de a prezenta modul în care se configurează `init`, trebuie descrisă noțiunea de **nivel de rulare** (*runlevel*). Un nivel de rulare se referă la un context de utilizare a unui sistem de operare. Altfel spus, se referă la ce servicii oferă sistemul utilizatorului. Un *runlevel* diferă de un altul prin dispozitivele montate și prin serviciilor pornite.

Sunt definite 7 niveluri de rulare, numerotate de la 0 la 6. Acestea pot fi folosite diferit de diversele distribuții Linux, însă 3 dintre ele au tot timpul aceeași semnificație:

- **Runlevel 0** – *Halt*: este folosit pentru oprirea sistemului;
- **Runlevel 1** – `Single`: este folosit pentru un sistem single-user;
- **Runlevel 6** – `Reboot`: este folosit pentru repornirea sistemului.

Pentru afișarea *runlevel*-ului curent se folosește comanda **`runlevel`** (ca root) sau **`who -r`** (ca utilizator obisnuit):

```
1 root@asgard:~# runlevel
2 N 2
3
4 razvan@asgard:~$ who -r
5          run-level 2  2009-09-23 17:57                   last=
```

Schimbarea runlevel-ului se realizează prin rularea comenzii **`init`** urmată de numărul runlevel-ului. Astfel rularea comenzii

```
root@asgard:~# init 0
```

este echivalentă cu rularea comenzii:

```
root@asgard:~# shutdown -h
```

sau

```
root@asgard:~# halt
```

Configurarea init

Fișierul principal de configurare pentru `init` este `/etc/inittab`. Este prezentată, în continuare, o parte a unui fișier de configurare `/etc/inittab` pentru a descrie modul în care acesta afectează comportamentul `init`:

```
[...]
# The default runlevel.
id:2:initdefault:
[...]

# /etc/init.d executes the S and K scripts upon change
# of runlevel.
#
# Runlevel 0 is halt.
# Runlevel 1 is single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.

l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
# Normally not reached, but fallthrough in case of emergency.
z6:6:respawn:/sbin/sulogin

# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
[...]
# /sbin/getty invocations for the runlevels.
#
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
#
# Format:
#  <id>:<runlevels>:<action>:<process>
#
# Note that on most Debian systems tty7 is used by the X Window System,
# so if you want to add more getty's go ahead but skip tty7 if you run X.
#
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
```

```
40 3:23:respawn:/sbin/getty 38400 tty3
41 4:23:respawn:/sbin/getty 38400 tty4
42 5:23:respawn:/sbin/getty 38400 tty5
43 6:23:respawn:/sbin/getty 38400 tty6
```

După cum se poate observa intrările din `/etc/inittab` au următorul format:

```
1 <id>:<runlevels>:<action>:<process>
```

unde:

- `id` este un identificator unic al regulii din inittab;
- `runlevels` reprezintă nivelurile de rulare asupra cărora se aplică regula;
- `action` este acțiunea pe care o va executa init;
- `process` este comanda care va fi executată.

O descriere completă a regulilor posibile pentru `inittab` se găsește în pagina de manual (**`man inittab`**). Vor fi prezentate, în continuare, regulile din exemplul de mai sus.

Linia 3 din captura de mai sus stabilește *runlevel*-ul implicit, care este `2`.

Identificatorii `10` până la `16` de pe liniile 14-20 prezintă reguli pentru scripturile de inițializare a serviciilor sistemului conform cu *runlevel*-ul. După cum se observă, comanda executată pentru fiecare *runlevel* în parte este `/etc/init.d/rc` urmată de numărul nivelului de rulare. Prin intermediul acestei comenzi se vor porni serviciile specifice sistemului, după cum se va vedea în continuare.

Linia cu identificatorul `ca` (linia 25) precizează că se execută o repornire a sistemului în cazul în care se apasă `CTRL-ALT-DEL`. Pentru această repornire se rulează comanda **`shutdown`**.

Regulile cu identificatorii 1 până la 6 (liniile 38-43) sunt folosite pentru a porni procesul `getty` pe terminalele virtuale ale sistemului. După cum se observă și din comentariu, terminalul virtual 7 este asociat mediului grafic și nu poate fi folosit pentru a porni `getty`. Acțiunea `respawn` semnifică faptul că, în cazul terminării procesului, `init` îl va recrea.

Pornirea daemonilor de sistem

După cum s-a observat din fișierul de configurare `/etc/inittab`, pornirea serviciilor sistemului în funcție de *runlevel* se realizează cu ajutorul scriptului `/etc/init.d/rc`. Acest script primește ca argument numărul *runlevel*-ului și execută scripturile aflate într-un director specializat: `/etc/rcX.d/`, unde `X` este numărul *runlevel*-ului. Astfel, dacă *runlevel*-ul este 3, se vor executa script-urile din `/etc/rc3.d/`.

Conținutul directorului `/etc/rc3.d/` este:

```
1 razvan@asgard:~$ ls -l /etc/rc3.d/
2 total 4
3 lrwxrwxrwx 1 root   root    18 2009-06-10 22:56 K08vmware -> /etc/init.d/
  vmware
4 lrwxrwxrwx 1 root   root    20 2009-07-22 14:17 K77ntp-server -> ../init.
  d/ntp-server
5 [...]
```

```
6  lrwxrwxrwx 1 razvan razvan  15 2009-01-17 11:54 S20exim4 -> ../init.d/
   exim4
7  lrwxrwxrwx 1 razvan razvan  19 2009-06-01 12:51 S20firebird2 -> ../init.d
   /firebird2
8  [...]
9  lrwxrwxrwx 1 root   root    18 2009-06-10 22:56 S90vmware -> /etc/init.d/
   vmware
10 lrwxrwxrwx 1 razvan razvan  17 2009-02-24 00:05 S91apache2 -> ../init.d/
   apache2
11 lrwxrwxrwx 1 razvan razvan  18 2009-01-17 11:47 S99rc.local -> ../init.d/
   rc.local
12 lrwxrwxrwx 1 razvan razvan  19 2009-01-17 11:47 S99rmnologin -> ../init.d
   /rmnologin
13 lrwxrwxrwx 1 razvan razvan  23 2009-01-17 11:47 S99stop-bootlogd -> ../
   init.d/stop-bootlogd
```

Se observă că fișierele sunt un set de legaturi simbolice către intrări din `/etc/init.d/`. Directorul `/etc/init.d/` conține scripturile de pornire/oprire/repornire a serviciilor din sistem. Astfel, pentru oprirea, pornirea sau repornirea serviciului `apache`, se folosesc comenzile:

```
1 root@asgard:~# /etc/init.d/apache start|stop|restart
```

Argumentele `start` / `stop` / `restart` sunt comune majorității scripturilor ce se găsesc în `/etc/init.d`.

De asemenea, se poate observa și un format al numelor fișierelor din `/etc/rcX.d`:

```
1 K numar nume
2 S numar nume
```

`K` înseamnă *kill*, iar `S` înseamnă *start*. Astfel, existența legăturii `K08vmware` înseamnă că serviciul `vmware` va fi **oprit** (prin intermediul comenzii `/etc/init.d/vmware stop`) odată cu intrarea în nivelul de rulare 3. La fel, existența legăturii `S20exim4` înseamnă că serviciul `exim` va fi **pornit** (prin intermediul comenzii **`/etc/init.d/exim start`**) odată cu intrarea în nivelul de rulare 3.

Numărul asociat unui din formatul numelor de fișiere reprezintă ordinea de pornire/oprire a serviciilor. Astfel, serviciul `S91apache` va fi pornit înaintea `S99rmnologin`, iar serviciul `K77ntp-server` va fi oprit înaintea `K08vmware` (la oprire ordinea este descrescătoare).

sysv-rc-conf

Avantajul schemei `init` de pornire a serviciilor este flexibilitatea, iar dezavantajul – complexitatea crescută și un număr mare de fișiere de configurare. Dacă se dorește eliminarea sau adăugarea unui serviciu la un *runlevel*, va trebui creată o legătură simbolică către scriptul de pornire/oprire și denumită folosind schema specifică. Procesul poate fi anevoios în cazul în care se dorește o configurare mai avansată.

Pentru a elimina acest deficient, se poate folosi utilitarul **`sysv-rc-conf`**. Acesta permite adăugarea/eliminarea facilă a unui serviciu dintr-un runlevel.

`sysv-rc-conf` oferă două interfețe de configurare a serviciilor: una în linia de comandă și una text. Interfața text permite parcurgerea serviciilor existente în sistem și adăugarea/elimitarea acestora dintr-un *runlevel* printr-o simplă acțiune de

check/*uncheck* (folosind tasta `SPACE`). Interfața în linia de comandă permite adăugarea/eliminarea unui serviciu prin precizarea nivelului/nivelurilor de rulare dorite, a numelui serviciului și a acțiunii (*on* sau *off*). În exemplul de mai jos, serviciul `postfix` a fost adăugat la nivelurile de rulare 4 și 5 și serviciul `apache` a fost eliminat din nivelurile de rulare 2 și 5.

```
root@anaconda:/boot/grub# sysv-rc-conf --list postfix
postfix      0:off      1:off   2:on    3:off   4:off   5:off   6:off

root@anaconda:/boot/grub# sysv-rc-conf --level 45 postfix on

root@anaconda:/boot/grub# sysv-rc-conf --list postfix
postfix      0:off      1:off   2:on    3:off   4:on    5:on    6:off

root@anaconda:/boot/grub# sysv-rc-conf --list apache
apache       0:off      1:off   2:on    3:on    4:on    5:on    6:off

root@anaconda:/boot/grub# sysv-rc-conf --level 25 apache off

root@anaconda:/boot/grub# sysv-rc-conf --list apache
apache       0:off      1:off   2:off   3:on    4:on    5:off   6:off
```

6.5.2 upstart

Începând cu versiunea 6.10 (Edgy Eft), distribuția Ubuntu a înlocuit `init` cu **upstart**. upstart permite o configurare asincronă, bazată pe evenimente care au loc în sistem. Spre deosebire de alte alternative pentru `init`, upstart este compatibil cu acesta și poate folosi scripturile `init` nemodificate (spre exemplu cele din `/etc/init.d`). upstart este, de fapt, un daemon `init` modificat. Ca orice sistem Linux, upstart este pornit prin rularea `/sbin/init`.

În septembrie 2009 echipa de dezvoltare a distribuției Debian a anunțat faptul că va renunța la init în favoarea upstart[1].

Upstart a apărut ca urmare a dezvoltării hardware-ului. Dacă până de curând se putea ști din momentul pornirii sistemului ce dispozitive există în sistem sau unde se poate monta sistemul de fișiere, dezvoltarea de noi dispozitive *hot-plug* (care pot fi conectate sau deconectate în timp real în calculator) a impus folosirea unui nou sistem de detecție și configurare a acestora.

upstart este așadar un daemon init bazat pe evenimente (*event-based init daemon*). Funcționarea sa înseamnă crearea de noi sarcini pentru sistemul de operare în două moduri:

- rularea de scripturi sau executabile în momentul producerii unui eveniment;
- oprirea sau pornirea unui serviciu în momentul producerii unui eveniment.

Un exemplu este montarea unui sistem de fișiere (spre exemplu de pe un USB *flash drive*). Pașii urmați, după conectarea dispozitivului USB, sunt:

- un eveniment de tip *startup* pornește daemon-ul `udevd`;

[1] http://lwn.net/Articles/351013/

- daemon-ul `udevd` este configurat să trimită un eveniment de tipul *block-device-added* pentru fiecare dispozitiv nou conectat;
- evenimentul *block-device-added* determină verificarea dispozitivului și montarea acestuia dacă este listat în `/etc/fstab`;
- după montarea sistemelor de fișiere din `/etc/fstab` se transmite evenimentul *fhs-filesystem*.

Configurare upstart

upstart este configurat prin intermediul directorului `/etc/event.d/` care se dorește a fi un înlocuitor pentru scripturile `/etc/inittab`, `/etc/init.d/*` și `/etc/rcX.d`. Fiecare fișier din `/etc/event.d` descrie modul în care un job trebuie coordonat:

```
root@ubuntui:/etc/event.d# ls
control-alt-delete  rc0  rc3  rc6          sulogin  tty3  tty6
logd                rc1  rc4  rcS          tty1     tty4
rc-default          rc2  rc5  rcS-sulogin  tty2     tty5
```

Spre exemplu, fișierul `control-alt-delete` prezintă acțiunea întreprinsă la apăsarea `CTRL-ALT-DEL`, și anume repornirea sistemului:

```
root@ubuntui:/etc/event.d# cat control-alt-delete
# control-alt-delete - emergency keypress handling
#
# This task is run whenever the Control-Alt-Delete key combination is
# pressed.  Usually used to shut down the machine.

start on control-alt-delete

exec /sbin/shutdown -r now "Control-Alt-Delete pressed"
```

Scripturile `rc*` sunt aociate cu nivelurile de rulare. După cum se observă, la fel ca în `/etc/inittab`, se apelează `/etc/init.d/rc` cu argument numărul nivelului de rulare:

```
root@ubuntui:/etc/event.d# cat rc2
[..]
start on runlevel 2
stop on runlevel [!2]
script
[...]
        exec /etc/init.d/rc 2
end script
```

Scripturile `tty*` prezintă modul de pornire a getty pentru diversele terminale virtuale (la fel ca în `/etc/inittab`).

Începând cu Ubuntu 7.04 (Feisty Fawn), upstart a înlocuit câteva procese daemon importante cum sunt `cron`, `atd`, `anacron` și `inetd`.

6.5.3 getty și login

După pornirea daemonilor sistemului, `init` este responsabil cu pornirea proceselor care asigură autentificarea utilizatorului în sistemul de calcul. După cum s-a văzut în cadrul fișierului de configurare `/etc/inittab`, `init` pornește câte un proces getty pentru fiecare terminal virtual pe baza imaginii `/sbin/getty`. În mod implicit sunt pornite 6 procese getty pentru terminalele denumite `tty1`, `tty2`, ..., `tty6`. Terminalul `tty7` este rezervat mediului grafic (în cazul în care acesta este instalat). Autentificarea în mediul grafic (prin introducerea unui utilizator/parolă) nu se realizează de către getty, ci de *display manager* (vezi secțiunea 13.2.2), o componentă a sistemului de ferestre.

Terminalele virtuale sunt accesate prin intermediul combinației de taste `CTRL-ALT-Fn`, unde `n` este numărul terminalului. Combinația `CTRL-ALT-F7` este folosită pentru accesarea mediului grafic.

getty este responsabil cu crearea unei interfețe de autentificare a utilizatorului și este responsabil atât pentru autentificarea prin terminale virtuale cât și pentru conexiunea pe alte terminale precum interfețele seriale.

După inițializarea terminalului, getty oferă prompt-ul "`Login:`" către utilizator pe terminalul virtual. Utilizatorul va trebui să introducă numele de utilizator și să apese `ENTER`. După introducerea numelui de utilizator, getty invocă `/bin/login`. Astfel, imaginea sa este înlocuită cu imaginea `/bin/login` și se creează procesul `login`.

În continuare este cerută parola utilizatorului (ale cărei caractere nu vor fi afișate pe ecran în momentul tastării lor din motive de siguranță). În acest moment, `login` poate verifica validitatea utilizatorului care a dorit să se autentifice. Dacă utilizatorul a introdus o pereche nume de utilizator/parolă corectă atunci, în mod tipic, i se va deschide o sesiune de shell pe terminalul curent. Altfel, procesul login moare, și, după un interval de așteptare, getty este repornit și oferă din nou promptul de autentificare utilizatorului.

Schema de autentificare și pornire a sesiunii de shell (în cazul unei autentificări reușite), este prezentată în figura 6.8:

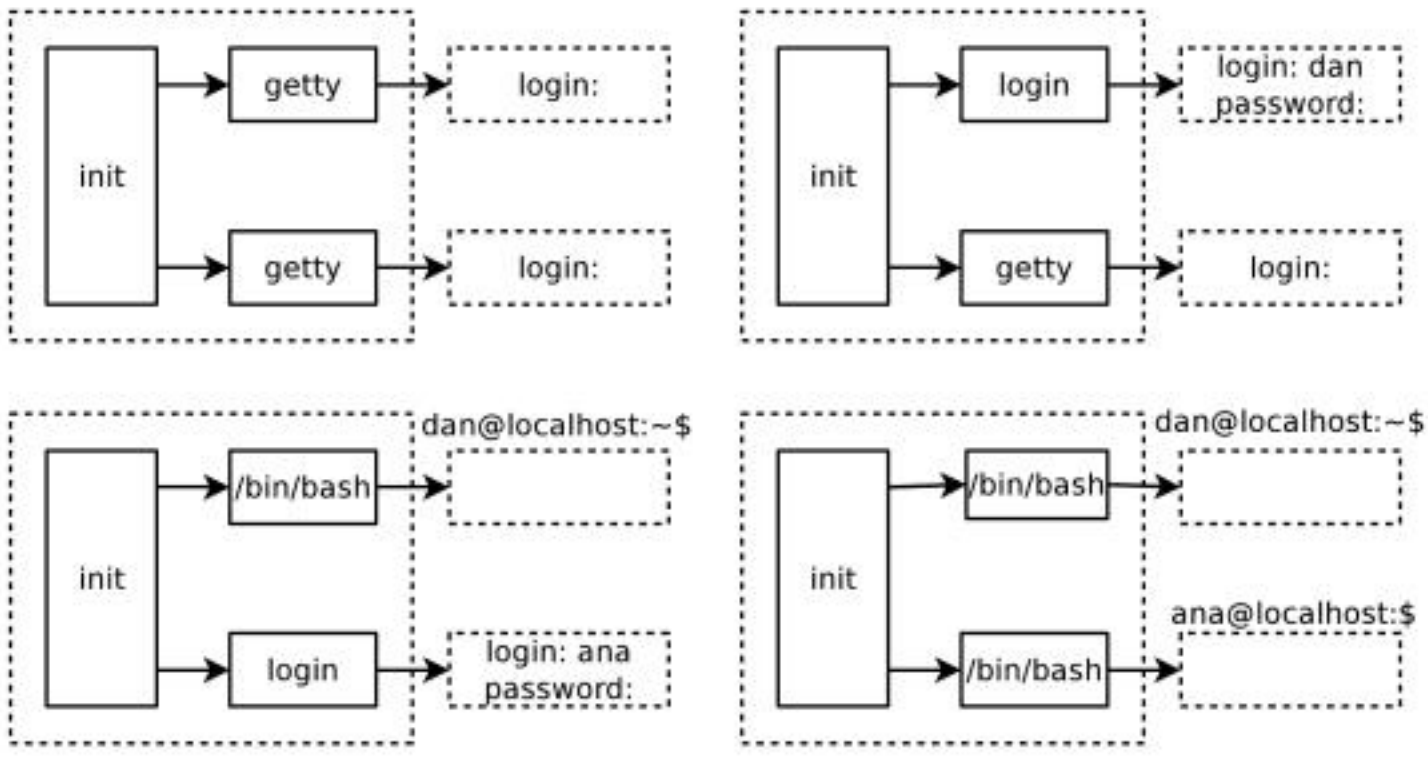


Figura 6.8: Autentificarea într-un sistem Linux

Se observă că se rulează getty pe toate terminalele. În momentul în care utilizatorul

introduce numele de utilizator, getty crează procesul login care cere parola utilizatorului. O dată cu introducerea și validarea parolei, utilizatorului îi este permis accesul în sistem și se deschide, în mod tipic, o sesiune shell. După cum se observă în figura 6.8, utilizatorul `dan` rulează shell-ul Bash (`/bin/bash`).

La încheierea sesiunii de shell curente, procesul getty este reponit așa cum este configurat în `/etc/inittab` (*respawn*).

Fișierele /etc/passwd și /etc/shadow

Fișierele `/etc/passwd` și `/etc/shadow` sunt utilizate pentru autentificarea utilizatorilor în sistem și stocarea de informații despre aceștia. După cum se poate observa dintr-o analiză simplă a fișierului `/etc/passwd`:

```
1 razvan@asgard:~$ cat /etc/passwd | grep guest
2 guest:x:1001:1001:Guest Account,,,:/home/guest:/bin/bash
```

O intrare din acest fișier este de forma:

```
1 username:simbol_parola:uid:gid:extra_info:home_dir:interpretor
```

Două dintre elementele prezentate sunt folosite de login pentru autentificarea utilizatorilor: `simbol_parola` și `interpretor`.

Câmpul `interpretor` specifică ce comandă va fi executată în momentul în care un utilizator a fost autentificat cu succes în sistem. De obicei această comandă este un interpretor de comenzi (în exemplul de mai sus `/bin/bash`), dar poate fi configurată orice altă comandă. În figura 6.8 utilizatorul `dan` folosește `/bin/bash`. Dacă se va completa intrarea asociată cu `/usr/bin/vim`, atunci orice autentificare pe un terminal virtual va rezulta în deschiderea editorului `vim` pe acel terminal.

Al doilea câmp dintr-o intrare în `/etc/passwd` (denumit aici `simbol_parola`) era folosit inițial pentru a stoca parola într-un format criptat. În felul acesta, oricine avea acces la fișier avea acces la parola criptată și putea folosi diverși algoritmi pentru a o sparge. O soluție ar fi fost restricționarea accesului la fișier. Totuși, acesta conține și alte informații necesare altor comenzi. Soluția aleasă a fost crearea unui nou fișier, `/etc/shadow`, care poate fi accesat doar de utilizatorul privilegiat.

```
1 root@anaconda:~# ls -l /etc/shadow
2 -rw-r----- 1 root shadow 1084 2009-07-21 22:19 /etc/shadow
```

Fișierul shadow conține numele de utilizator și parola criptată. Pentru mai multe detalii consultați paginile de manual (**`man 5 passwd`**, **`man 5 shadow`**) sau secțiunea 10.2.3.

6.5.4 Sesiune de shell

După rularea `init` și autentificarea utilizatorului în sistem, acestuia îi este oferită, în mod tipic, o sesiune de shell (conform configurației din fișierul `/etc/passwd`).

Înaintea oferirii promptului către utilizator, procesul shell nou creat citește un set de fișiere de configurare. Acestea sunt folosite pentru a stabili formatul promptului, umask-ul curent, aliasurile de comenzi, variabilele de mediu.

În cazul unui interpretor bash (*Bourne Again Shell*), fișierele de configurare citite sunt în ordine:

- `/etc/profile`: acesta este fișierul global de configurare pentru sesiunile de shell;
- `~/.bash_profile` sau `~/.bash_login` sau `~/.profile`: se execută primul găsit;
- `~/.bashrc` este fișierul obișnuit de configurare locală; drept urmare, `~/.bash_profile` conține linia

```
1 if [ -f ~/.bashrc ]; then . ~/.bashrc; fi
```

pentru a interpreta acest fișier.

După interpretarea fișierelor de configurare, se stabilește mediul curent de lucru și utilizatorului îi este oferit promptul unde poate introduce comenzi.

6.6 Studiu de caz

6.6.1 Grub 2

În ciuda faptului că nu a oferit nici o îmbunatățire majoră în ultimii ani, GRUB a rămas cel mai utilizat *bootloader* în cadrul distribuțiilor GNU/Linux. Aceasta datorită faptului că *bootloader*-ul este una din cele mai delicate componente ale sistemului, o actualizare greșită a sa ducând la nefuncționarea sistemului de operare.

Unul din elementele care au contribuit la lipsa de îmbunatățiri aduse GRUB este și arhitectura veche și greu de extins. În 2002 a început lucrul la o nouă versiune de GRUB (GRUB2), versiune care a adus aducă schimbări destul de mari.

GRUB2[1][2] a fost rescris de la zero pentru a asigura o portabilitate și o modularitate superioară versiunii anterioare. Printre avantajele pe care le oferă se numără:

- suport pentru scripturi;
- portabilitate pe mai multe arhitecturi;
- suport pentru i18n[3];
- un nou sistem de denumire a partițiilor.

Fișiere de configurare

GRUB2 se configurează într-un mod diferit față de GRUB. În primul rând numele fișierului de configurare s-a schimbat din `/boot/grub/menu.lst` în `/boot/grub/grub.cfg`. Spre deosebire de GRUB, noul fișier de configurare nu mai este editat direct de către utilizator, ci este generat automat:

[1] http://www.gnu.org/software/grub/grub-2.en.html
[2] https://wiki.ubuntu.com/Grub2
[3] http://en.wikipedia.org/wiki/Internationalization_and_localization

```
root@ant:~# cat /boot/grub/grub.cfg
#
# DO NOT EDIT THIS FILE
#
# It is automatically generated by /usr/sbin/grub-mkconfig using
templates
# from /etc/grub.d and settings from /etc/default/grub
#
[...]
```

Configurarea GRUB2 se realizează din trei locuri:

- fișierul `/etc/default/grub`: conține un set general de configurări, legate în principal de diverse opțiuni ale meniului de boot;
- directorul `/etc/grub.d`: conține un set de scripturi care descoperă imaginile de kernel instalate în sistem și generează `grub.cfg`;
- directorul `/boot/grub`: conține *bootloader*-ul și diversele module pe care acesta le folosește.

După editarea unuia din fișierele din `/etc/grub.d` sau a `/etc/default/grub` va trebui rulată comanda **`update-grub2`** pentru ca fișierul `/boot/grub/grub.cfg` să fie regenerat.

Editarea opțiunilor meniului

Fișierul `/etc/default/grub` conține configurări ale meniului de boot. Spre exemplu `GRUB_DEFAULT` specifică opțiunea din meniu selectată în mod implicit, iar `GRUB_TIMEOUT` specifică *timeout*-ul după care opțiunea implicită va fi utilizată pentru bootare.

```
root@ant:~# cat /etc/default/grub
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.

GRUB_DEFAULT=0
GRUB_TIMEOUT="10"
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""
[...]
```

Adăugarea unei noi opțiuni în meniu de boot

Opțiunile din meniul GRUB2 sunt create folosind scripturile din directorul `/etc/grub.d`:

- `00_header`: încarcă informațiile din `/etc/default/grub`;
- `05_debian_theme`: configurează fundaluri, culori și teme;
- `10_linux`: caută imagini de kernel pe disc și le introduce în meniu;

- `20_memtest86+`: introduce în meniu imaginea `/boot/memtest86+.bin`, dacă aceasta există;
- `30_os-prober`: caută alte imagini de sisteme de operare și le include în meniu;
- `40_custom`: permite utilizatorului să introducă intrări statice în meniu.

Ordinea în care apar opțiunile în meniu este dată de ordinea numelor fișierelor. Astfel, întotdeauna `memtest86+` (`20_memtest86+`) va apărea după imaginile de kernel de pe disc (`10_linux`).

Pentru a adăuga o nouă opțiune există două posibilități: se editează fișierul `40_custom` sau se poate crea un fișier nou, caz în care numele fișierului nou creat va defini poziția intrării în meniu.

Un exemplu de sintaxă care se folosește pentru adăugarea de noi opțiuni este:

```
1  echo "Adding Custom Kernel \& SystemRescue" >\&2
2  cat << EOF
3  menuentry "Ubuntu, linux 2.6.31-11-custom" {
4          set root=(hd0,9)
5          linux /boot/vmlinuz-2.6.31-11-custom root=UUID=c6829e27-2350-4e84
   -bdbb-91b83f018f98 ro
6          initrd /boot/initrd.img-2.6.28-11-generic
7  }
8
9  menuentry "Boot SystemRescue CD from hard drive" {
10         set root=(hd1,10)
11         linux   /sysrcd/rescuecd subdir=sysrcd setkmap=us
12         initrd  /sysrcd/initram.igz
13 }
14 EOF
```

Linia `echo "Adding Custom Kernel & SystemRescue" >&2` nu este neapărat necesară. Dacă este inclusă, atunci textul *Adding...* va fi afișat în timpul execuției comenzii **`update-grub2`**. Celelalte opțiuni sunt similare cu cele din GRUB.

Unul din elementele de noutate pe care le-a introdus GRUB2 este o nouă schema de denumire a partițiilor. Astfel, discurile sunt denumite în forma `hdX`, unde X este un numar ce începe de la 0 (spre exemplu, primul disc SATA, este denumit `hd0`, al doilea disc SDB este `hd1` etc.), similar cu GRUB. În schimb, partițiile se denumesc de forma `sdaY`, cu Y începând de la 1 (de exemplu a treia partiție de pe sda este `sda3`).

6.6.2 Pornirea sistemului de operare Windows XP

Pornirea sistemului de operare Windows XP urmărește aceeași pași ca în cazul pornirii Linux: pornirea sistemului, încărcarea BIOS, încărcarea *bootloader*-ului, încărcarea nucleului, pornirea serviciilor.

Pașii până la încărcarea bootloader-ului sunt independenți de sistemul de operare și, deci, identici cu cei pentru Linux.

Bootloader-ul pentru Windows NT (NT, 2000, XP, 2003) este **NT Loader** (NTLDR).

NTLoader

La fel ca și în cazul Linux, sectorul de boot al hard-disk-ului (MBR) sau al unei partiții (VBR) este limitat și este folosit pentru încărcarea NTLDR. Imaginea acestuia rezidă în `C:\Windows\ntldr`.

După încărcarea bootloader-ului, acesta consultă fișierul `boot.ini` pentru a localiza informațiile de pe partiția de sistem.

Configurare NTLoader (boot.ini)

Fișierul `boot.ini` se află pe partiția de sistem (`C:`). Pentru vizualizarea și accesarea acestuia trebuie permisă vizualizarea fișierelor de sistem (`Tools->Folder Options ->View->Hide protected operating system files`).

Un exemplu de conținut al `boot.ini` este prezentat în continuare:

```
1 [boot loader]
2 timeout=30
3 default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
4 [operating systems]
5 multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP
  Professional" /noexecute=optin /fastdetect
```

În cazul existenței mai multor sisteme Windows, utilizatorului îi va fi afișat un ecran de selecție (asemănător cu cel al GRUB).

Variabila `timeout` specifică timpul de așteptare înainte de a încărca sistemul de operare dat de opțiunea implicită.

Lista cu opțiunile posibile de boot-are a nucleului Windows XP se poate consulta aici[1].

Încărcarea nucleului

După rularea *bootloader*-ului, acesta încarcă în memorie imaginea de nucleu a Windows, `ntoskrnl.exe`, și `hall.dll` (*Hardware Abstraction Layer*), controlul fiind apoi oferit nuclelului.

Nucleul este încărcat în memorie în două faze: în prima fază se inițializează dispozitivele sistemului (procesor, memorie), iar în a doua fază se încarcă device driver-ele de sistem necesare.

Pornirea proceselor importante

După încărcarea nucleului în memorie, sunt pornite, în ordine, două procese importante:

- **smss** (*Session Manager Subsystem*): acesta montează dispozitivele din sistem, creează variabilele de mediu și pornește winlogon (*Windows Logon Manager*);

[1] http://support.microsoft.com/default.aspx?scid=kb;en-us;833721

- **winlogon** (*Windows Logon Manager*): este procesul care se ocupă de autentificarea utilizatorilor în sistem; în urma unei autentificări reușite sunt pornite shell-ul Windows (`explorer`) și alte procese ale utilizatorului.

Mai multe detalii despre procesul de pornire a unui sistem Windows XP se pot găsi aici[1].

6.6.3 Interoperabilitate Linux-Windows

În cazul sistemelor dual-boot Linux-Windows se poate întampla să se suprascrie MBR-ul și să se piardă astfel opțiunea directă de boot-are a unuia dintre sistemele de operare. Pentru a recupera *bootloader*-ul pierdut, trebuie rescris MBR-ul.

Reinstalarea GRUB

Situația cea mai frecventă ce necesită o reinstalare GRUB este aceea în care se reinstalează Windows pe un sistem pe care există instalată și o distribuție Linux. La instalarea Windows suprascrie MBR astfel încât se pierde *stage1* din GRUB.

Pentru a reinstala GRUB, este necesar un mediu boot-abil Linux. Acesta poate fi un LiveCD sau o dischetă boot-abilă. În cazul unei dischete boot-abile utilizatorului îi este oferit direct prompt-ul de grub. În cazul unui LiveCD, utilizatorul va trebui să deschidă o consolă și să acceseze de acolo consola de GRUB:

```
root@asgard:~# grub
Probing devices to guess BIOS drives. This may take a long time.

    GNU GRUB  version 0.97  (640K lower / 3072K upper memory)

       [ Minimal BASH-like line editing is supported.  For
         the  first  word,  TAB  lists  possible  command
         completions.  Anywhere else TAB lists the possible
         completions of a device/filename. ]

grub>
```

Presupunând că se dorește instalarea GRUB pe primul hard-disk din sistem, vor trebui urmați pașii de mai jos:

- specificarea partiției care conține imaginea `stage2` de GRUB:

```
grub> root (hd0,2)
```

- dacă nu se știe pe ce partiție se află imaginile se poate folosi comanda **`find`**, căutând unul din fișierele din `/boot`. Cel mai adesea este căutat `stage1`:

```
grub> find /boot/grub/stage1
 (hd0,2)
```

- după descoperirea partiției se realizează pasul 1, apoi se instalează imaginea de GRUB în MBR:

```
grub> setup (hd0)
```

[1] http://technet.microsoft.com/en-us/library/bb457123.aspx

- imaginea de GRUB poate fi instalată și în sectorul de boot al unei partiții (VBR):

```
grub> setup (hd0,0)
```

  însă, în acest caz, GRUB va trebui să fie pornit (*chainloaded*) cu un alt bootloader.

Comanda **`quit`** se folosește pentru ieșirea din consola GRUB. După repornirea sistemului, acesta va afișa ecranul de opțiuni GRUB pentru selecția sistemului de operare dorit.

Reinstalarea NTLoader

Reinstalarea NTLDR este necesară în cazul în care se instalează o distribuție Linux fără a se instala și un bootloader (spre exemplu GRUB), sau în cazul unui MBR corupt.

Pentru repararea MBR, trebuie folosită consola de recuperare Windows XP (*Recovery Console*). Pentru accesarea acesteia se folosește CD-ul de instalare Windows și după bootare se alege opțiunea *Repair*. În continuare se apasă `R` pentru rularea *Emergency Repair Process* și se alege *Fast* sau *Manual Repair*. *Fast Repair* este recomandată majorității utilizatorilor, iar *Manual Repair* administratorilor și utilizatorilor avansați.

Mai multe detalii găsiți online[1].

6.6.4 Personalizarea ecranului GRUB

Ecranul de opțiuni prezentat de GRUB poate fi anost. Pentru a rezolva acest neajuns se poate folosi o altă imagine de fundal, denumită GRUB *splashimage*. Există numeroase site-uri care oferă astfel de imagini[2][3][4]). Imaginile sunt imagini .xpm arhivate (de exemplu hubble.xpm.gz).

Pentru folosirea unei imagini splash de GRUB trebuie editat fișierul principal de configurare a GRUB (`/boot/grub/menu.lst`). Pașii ce trebuiesc urmați sunt prezentați mai jos:

- crearea unei copii de siguranță a fișierului de configurare GRUB:

```
root@asgard:/boot/grub# cp menu.lst menu.lst.old
```

- obținerea unui fișier .xpm.gz care conține imaginea de splash folosită pentru GRUB:

```
root@asgard:/boot/grub# wget http://schragehome.de/splash/hubble.xpm
.gz
--2009-09-26 06:23:27--  http://schragehome.de/splash/hubble.xpm.gz
Resolving schragehome.de... 82.165.105.215
Connecting to schragehome.de|82.165.105.215|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 99593 (97K) [image/x-xpixmap]
Saving to: 'hubble.xpm.gz'
```

[1] http://www.ntfs.com/missing-corrupted-system-files.htm
[2] http://schragehome.de/splash/
[3] http://www.queervisions.com/arch/2007/04/9_grub_splash_i.html
[4] http://kde-look.org/index.php?xcontentmode=35x45

```
8
9  100%[=====================================>] 99,593       209K/s   in
   0.5s
10
11 2009-09-26 06:23:27 (209 KB/s) - 'hubble.xpm.gz' saved [99593/99593]
```

- editarea fișierului de configurare:

```
1 [...]
2 # Pretty colours
3 #color cyan/blue white/blue
4 splashimage=(hd0,2)/boot/grub/hubble.xpm.gz
5 [...]
```

  în exemplul de mai sus partiția unde se află imaginea de GRUB este a treia partiție de pe primul disc (`hd0,2`).

După editarea și salvarea fișierului se repornește sistemul. Noul ecran de opțiuni GRUB va folosi ca imagine de fundal `hubble.xpm.gz`.

Cuvinte cheie

- pornirea/repornirea unui sistem
- booting/bootstrapping
- fazele pornirii unui sistem de operare
- BIOS
- POST
- CMOS
- dispozitiv boot-abil
- sector de boot/sector bootabil
- MBR, VBR
- tabelă de partiții
- bootloader
- fazele unui bootloader (stages)
- GRUB
- stage1, *_stage1_5, stage2
- /boot/grub/menu.lst
- consola GRUB
- imagine de nucleu
- opțiuni de încărcare a nucleului
- init
- runlevel
- /etc/inittab
- /etc/init.d, /etc/rcX.d
- getty
- login
- /etc/passwd, /etc/shadow
- /etc/profile, ~/.bash_profile, ~/.bashrc
- NTLDR
- boot.ini
- smss, winlogon
- Recovery Console

Întrebări

1. Care este ordinea corectă a pornirii unui sistem?
   - ❑ vmlinuz, init, GRUB, BIOS
   - ❑ GRUB, BIOS, init, vmlinuz

- ❑ BIOS, GRUB, vmlinuz, init
- ❑ init, vmlinuz, BIOS, GRUB

2. Ce este un sector de boot?
   - ❑ primul sector al primei partiții
   - ❑ primul sector al unei partiții sau al unui hard-disk
   - ❑ un sector ce conține semnătura 0xAA55 (ultimii doi octeți)
   - ❑ un sector pe care se găsește faza a doua a GRUB (stage2)
3. Care din următoarele este un înlocuitor al init?
   - ❑ upstart
   - ❑ downstart
   - ❑ slowstart
   - ❑ faststart
4. Câte partiții primare, active, extinse, logice se pot afla pe un hard-disk?
   - ❑ oricâte, 1, 2, 4
   - ❑ 1, oricâte, oricâte, 4
   - ❑ 4, 1, 1, oricâte
   - ❑ 1, oricâte, 2, oricâte
5. Care din următoarele asocieri NU este validă?
   - ❑ GRUB/NTLDR
   - ❑ vmlinuz/ntoskrnl
   - ❑ POST/CMOS
   - ❑ getty/winlogon
6. Care fază a GRUB este resposabilă cu încărcarea imaginii de nucleu Linux?
   - ❑ stage1
   - ❑ stage1.5
   - ❑ stage2
   - ❑ stage3
7. Care este fișierul principal de configurare GRUB pe un sistem Debian/Ubuntu?
   - ❑ /etc/main.cf
   - ❑ /etc/grub.conf
   - ❑ /boot/grub.conf
   - ❑ /boot/grub/menu.lst
8. Care este ordinea corectă de pornire a unei sesiuni shell?

- ❑ login, init, getty, bash
- ❑ getty,bash, login, init
- ❑ init, getty, login, bash
- ❑ init, login, bash, getty

9. Cum se reprezintă, în notația GRUB, a 3-a partiție de pe al doilea disc?
   - ❑ /dev/hdb3
   - ❑ /dev/hda2
   - ❑ (hd1,2)
   - ❑ (hd2,3)
10. Fișierul `/etc/shadow` conține parolele criptate ale utilizatorilor sistemului. Poate fi accesat numai de utilizatori privilegiați ai sistemului.
    - ❑ adevărat, adevărat
    - ❑ adevărat, fals
    - ❑ fals, adevărat
    - ❑ fals, fals

Capitolul 7

Analiza hardware a sistemului

It's hardware that makes a machine fast. It's software that makes a fast machine slow.

Craig Bruce

Ce se învață din acest capitol?

- Principalele componente hardware ale unui sistem de calcul
- Mecanismele de accesare și interogare a dispozitivelor hardware
- Încărcarea si folosirea modulelor de kernel
- Utilitare de interacțiune cu dispozitivele: **`lspci`**, **`lsusb`**, **`smartmontools`**
- Lucrul cu dispozitivele sistemului; comanda **`dd`**
- Interacțiunea cu sistemul: `/proc`, **`sysctl`**
- Cum se poate realiza o copie de siguranță pentru MBR

7.1 Structura unui sistem de operare

Sistemul de operare este cel care se ocupă de managementul resurselor hardware și software prezente în computer. SO are mai multe componente, fiecare cu un rol bine definit. Diagrama 7.1 prezintă modul în care este structurat accesul la hardware.

Sistemul de operare este compus din:

- **kernel**: nucleul sistemului de operare – se ocupă de comunicația directă cu hardware-ul și oferă un sistem de management al resurselor pentru aplicațiile din sistem;
- **biblioteci de funcții** – pun la dispoziție aplicațiilor un API (*Application Programming Interface*) comun pentru accesul la resursele oferite de kernel;
- **aplicații** – cele care vin cu un sistem de operare sunt de obicei utilizate pentru configurarea sistemului, modificarea datelor din sistem etc. Aceste aplicații pot să utilizeze biblioteci de funcții sau pot comunica direct cu kernel-ul.

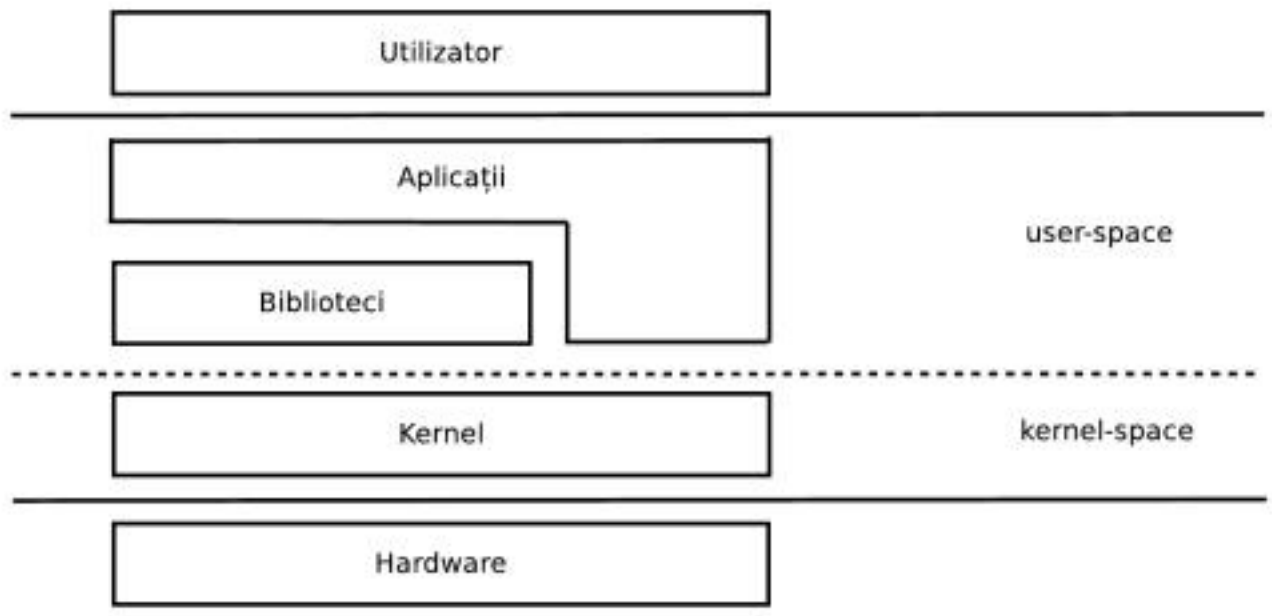


Figura 7.1: Accesul la hardware

Sistemul de operare face distincția între două "spații" în care se pot găsi programe în execuție: spațiul utilizator (*user-space*) și spațiul kernel (*kernel-space*). Prin "spațiu" se înțelege atât spațiu de memorie cât și mediu de execuție. Fiecare spațiu are alte condiții de lucru.

Spațiul kernel este spațiul în care rulează kernel-ul. Conține doar kernel-ul și componentele sale (precum driver-ele încărcate la un moment dat). Tot ceea ce se găsește în spațiul kernel are acces direct la hardware.

Spațiul utilizator este spațiul în care se găsesc aplicațiile și bibliotecile de funcții utilizate de acestea la un moment dat. Tot ceea ce se găsește în spațiul utilizator nu are acces direct la hardware.

Toate acțiunile aplicațiilor din spațiul utilizator trebuie să treacă prin kernel. Comunicația dintre spațiul utilizator și spațiul kernel se realizează prin apeluri de sistem.

Apelul de sistem este un mecanism prin care programele sau bibliotecile din spațiul utilizator accesează resursele puse la dispoziție de kernel.

Practic, apelurile de sistem sunt funcții implementate în kernel pe care aplicațiile din spațiul utilizator le accesează direct sau prin intermediul bibliotecilor de funcții.

Acest capitol va analiza modul de lucru intern pentru un sistem Linux.

7.2 Considerente hardware

Componentele fizice ale sistemului poartă denumirea generică de **hardware**. Pentru folosirea acestora, utilizatorul dispune de un sistem de operare și aplicații. Acestea reprezintă nivelul de software de peste hardware. Fiecare componentă este esențială și nu poate funcționa în lipsa celeilalte.

Sistemul de operare și aplicațiile de bază sunt cele care asigură interfața cu hardware-ul pentru a preveni, astfel, dificultatea programării directe a hardware-ului. Din punctul de

vedere al utilizatorului, interfața cu hardware-ul este limitată la un set de apeluri către sistemul de operare și aplicațiile de bază.

7.2.1 Structura unui sistem de calcul

Din punct de vedere structural, un sistem de calcul este un set de componente cu roluri bine definite care comunică pentru îndeplinirea unei sarcini. Sistemele de calcul din ziua de azi urmează structura von Neumann prezentată mai jos. Aceasta este folosită pentru a defini setul de componente care există într-un sistem de calcul și interacțiunea dintre acestea.

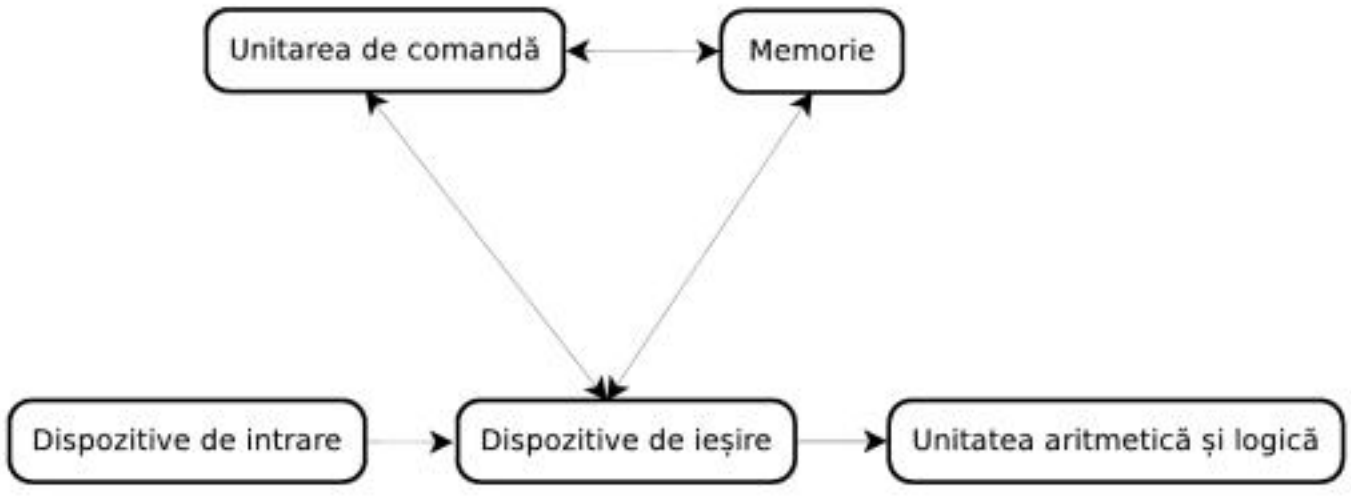


Figura 7.2: Interacțiunea componentelor hardware ale unui sistem de calcul

După cum se observă, arhitectura von Neumann precizează 5 componente ale unui sistem de calcul:

- **unitatea de comandă** sau **unitatea de control** este responsabilă cu comandarea modului în care celelalte componente lucrează; unitatea de comandă va interpreta codul unui program (va decodifica instrucțiunile) și va cere unității aritmetice și logice execuția sarcinilor necesare;

- **unitatea aritmetică și logică** se mai numește **unitatea de execuție**; rolul acesteia este de a executa codul transmis de la unitatea de comandă; unitatea aritmetică are un set de registre pentru stocarea locală a informației și un acumulator pentru efectuarea de operații; rolul unității aritmetice și logice este îndeplinit de procesor;

- **memoria** este componenta în care se stochează codul programului de executat și datele acestuia; memoria este folosită de unitatea de execuție și de comandă; unitatea de comandă citește din memorie instrucțiunile și le decodifică, pe când unitatea de execuție citește și scrie în memorie datele cu care efectuează diverse calcule;

- **dispozitivele de intrare/ieșire (periferice)** sunt cele care intermediază comunicația sistemului cu exteriorul; informațiile de la aceste dispozitive sunt copiate în memorie și folosite ulterior de unitatea de comandă și unitatea de execuție; exemple de dispozitive periferice sunt monitorul, tastatura, mouse-ul, hard disk-ul etc.

7.2.2 Procesorul

Procesorul sau unitatea centrală de prelucrare (CPU – *Central Processing Unit*) este "creierul" sistemului. Este echivalentul unității de comandă și a celei de execuție din arhitectura von Neumann. Procesorul este cel care va interpreta codul unui program și va efectua operații cu datele acestuia și este componenta de bază a execuției unui proces.

Procesorul este cel care determină arhitectura unui sistem de calcul. Astfel, procesoarele Intel sau AMD fac parte din arhitectura `x86`. Majoritatea procesoarelor mai noi fac parte din arhitectura `x86_64`. Arhitectura unui sistem de calcul se referă la setul de instrucțiuni pe care procesorul le oferă utilizatorului.

Procesorul dispune de un set de locații de stocare de foarte mare viteză denumite **registre**. Viteza procesorului este foarte mare comparativ cu viteza memoriei și de aceea este nevoie de registre pentru a efectua calcule rapide. În ziua de azi dimensiunea registrelor este de 32 sau de 64 de biți. Dimensiunea registrelor este sursa denumirii sistemului. Un sistem este denumit sistem pe 32 de biți sau sistem pe 64 de biți în funcție de dimensiunea registrelor sale.

Un procesor poate avea una sau mai multe unități de procesare (*core*). O unitate de procesare este un circuit electronic care execută instrucțiuni. Un procesor cu o singură unitate de procesare se numește **single-core**, iar un procesor cu două sau mai multe unități de procesare aflate pe același circuit integrat se numește **multi-core**. Performanța obținută prin folosirea mai multor unitati de procesare depinde de existența firelor de execuție multiple în cadrul aplicațiilor și de suportul sistemului de operare.

Caracteristica vitezei unui procesor este, de obicei, **frecvența** acestuia, măsurată în Hz. În zilele noastre, frecvența de lucru a unui procesor este de ordinul GHz. Calculatoarele persoanele folosesc procesoare cu arhitectură i386, precum: Intel Pentium, Intel Xeon, AMD Athlon, AMD Opteron etc.

Există un număr însemnat de arhitecturi menite să adreseze de la constrângeri specifice: spre exemplu cele pentru platforme integrate (embedded systems): ARM, MIPS, sau alte platforme de calcul intensiv: Alpha, PowerPC etc.

Pentru arhitecturi diferite poate varia numărul de biti pe care sunt efectuate operațiile sau numărul de registre aflate la dispoziția procesorului. Astfel nu putem compara viteza unui procesor i386 cu un procesor PowerPC folosind frecvența.

7.2.3 Ierarhia de memorie

În arhitectura unui calculator modern, memoria este organizată în mod ierarhic. Fiecare nivel din ierarhie are o lățime de bandă mai mare, dimensiune mai mică și latență mai scăzută decât nivelul inferior. După cum se observă în figura 7.3 cele 4 niveluri principale sunt: registrele procesorului, memoria cache, memoria RAM și hard disk-ul.

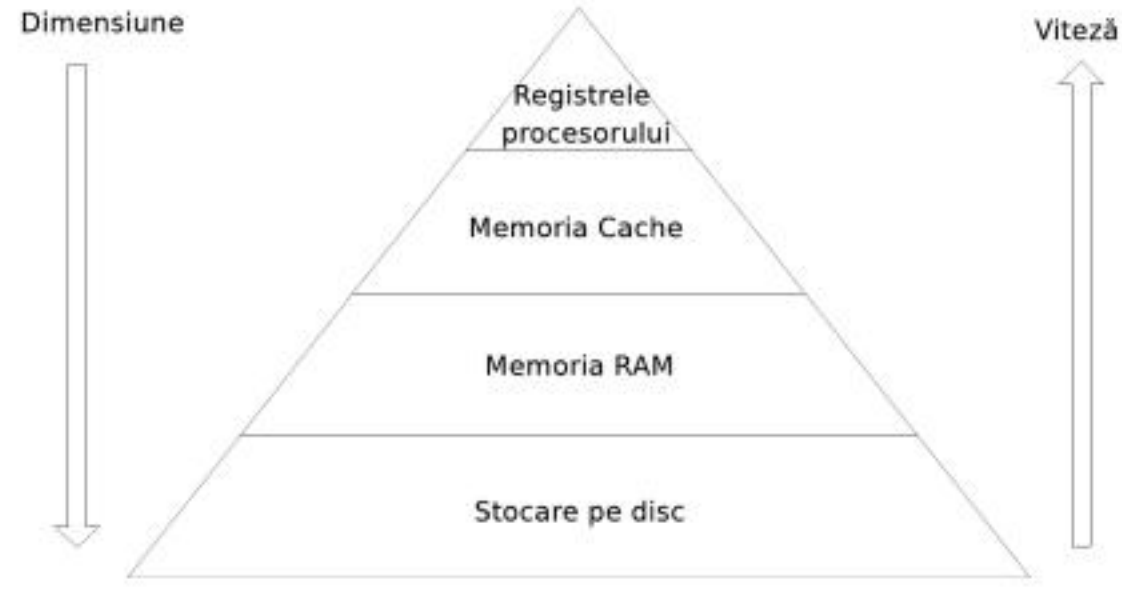


Figura 7.3: Ierarhia de memorie

Registrele procesorului

În vârful ierarhiei de memorii se află registrele procesorului care au cea mai mare viteză de acces, de obicei un ciclu de CPU, dar dimensiunea de câteva sute de bytes. Ele sunt folosite pentru a stoca datele care sunt prelucrate de către procesor la momentul curent.

Memoria cache

Din cauza diferenței mari de viteză între memoria RAM și procesor, se folosește un nivel intermediar de memorie: mai rapidă dar mai scumpă și de dimensiunea mai mică. Această memorie poartă numele de **memorie cache** și este folosită pentru reținerea anumitor informații din memoria RAM.

Memoria cache folosește conceptul de localizare temporală și spațială. Aceasta înseamnă că în momentul în care procesorul dorește să acceseze o zonă de memorie, mai întâi va compara adresa acesteia cu adresele zonelor de memorie care au fost copiate deja în cache până în acel moment. Dacă descoperă că această zonă există în cache, ea va fi accesată direct de acolo. În caz contrar, zona de memorie dorită va fi copiată în cache pentru a putea fi refolosită în momentele apropiate. Când cache-ul se umple, intrările din el sunt șterse în ordinea vechimii lor. În mod tipic, memoria cache are o dimensiune de ordinul sutelor de KB sau câtorva MB.

Există trei tipuri de memorie cache: L1, L2 și L3.

Memoria cache L1, sau de nivel 1, este o memorie încorporată în procesor, de dimensiune foarte mică dar foarte rapidă, fiind folosită pentru a accesa date importante și frecvent folosite de către procesor. Poate fi accesată în câtiva cicli de CPU și are dimensiunea de zeci de KB. Memoria cache L1 este de obicei de tipul SRAM (Static RAM), ceea ce înseamnă că nu necesită refresh pentru a menține datele.

Memoria cache L2 mai este numită și cache secundar, fiind folosită pentru a stoca date folosite recent. De obicei este externă procesorului dar se tinde spre încorporarea ei în

procesor. Poate fi folosită ca buffer pentru datele și instrucțiunile ce urmează a fi preluate din memoria RAM. Are o latență de 2 până la 10 ori mai mare decât memoria cache L1, dar dimensiunea mult mai mare. Inițial dimensiunea memoriei cache L1 era de 256 sau de 512 KB, iar în prezent are dimensiunea de ordinul MB.

Memoria cache L3 este o memorie specializată care cooperează cu memoria cache L1 și L2 pentru a îmbunătăți performanțele sistemului. Are dimensiunea mai mare decât memoriile L1 și L2, fiind de ordinul MB. Deși latența introdusă este mai mare decât pentru L1 și L2, accesarea ei este mai rapidă decât în cazul memoriei RAM.

Memoria RAM

Când ne referim la memoria unui sistem de calcul ne vom referi, de obicei, la memoria RAM (*Random Access Memory*). Aceasta este folosită pentru a stoca datele și codul unui proces. În momentul în care se dorește efectuarea unui calcul sau decodificarea unei instrucțiuni, se face un apel la memorie pentru citirea acelei informații.

În zilele noastre, memoria RAM folosită este de tipul DRAM (Dynamic RAM). Tipul cel mai recent este DDR3. Caracteristicile importante ale unui modul de memorie RAM sunt:

- frecvența de lucru: de ordinul MHz
- timpul de acces: de ordinul nanosecundelor
- rata de transfer: măsurată în GB/s
- capacitatea: actualmente de ordinul GB

Hard disk-ul

Probabil cel mai important dispozitiv periferic al sistemului este hard-disk-ul (HDD – *hard disk drive*). Acesta este folosit, de obicei, ca suport fizic pentru sistemele de fișiere. Hard-disk-urile reprezintă dispozitive de stocare permanentă (nepersistentă) a datelor.

Parametrii importanți care caracterizează un hard-disk sunt:

- capacitatea acestuia: în ziua de azi de ordinul sutelor de GB sau ordinul TB;
- viteza (măsurată în RPM – *Rotations Per Minute*): cu valori tipice de 5400 RPM, 7200 RPM, 10000 RPM;
- dimensiunea buffer-ului (un cache folosit pentru mărirea vitezei): cu valori de ordinul MB.

Există un set de interfețe și magistrale prin care un hard-disk poate fi accesat. Dintre acestea amintim ATA, Serial ATA, SCSI, SAS etc. Fiecare dintre aceste magistrale oferă o interfață distinctă de conectare a hard-disk-ului la placa de bază și un alt protocol de comunicare cu hard-disk-ul pentru accesarea informațiilor de pe acesta.

Un hard disk conține unul sau mai multe discuri neflexibile de metal numite platane. În cazul unui disc formatat, suprafața este împărțită în piste care sunt cercuri concentrice pe fiecare parte a platanelor. Pistele egal distantate de ax de pe fiecare parte a platanului

și de pe toate platanele, sunt grupate în cilindri. Fiecare pistă este împărțită în sectoare de câte 512 biți fiecare.

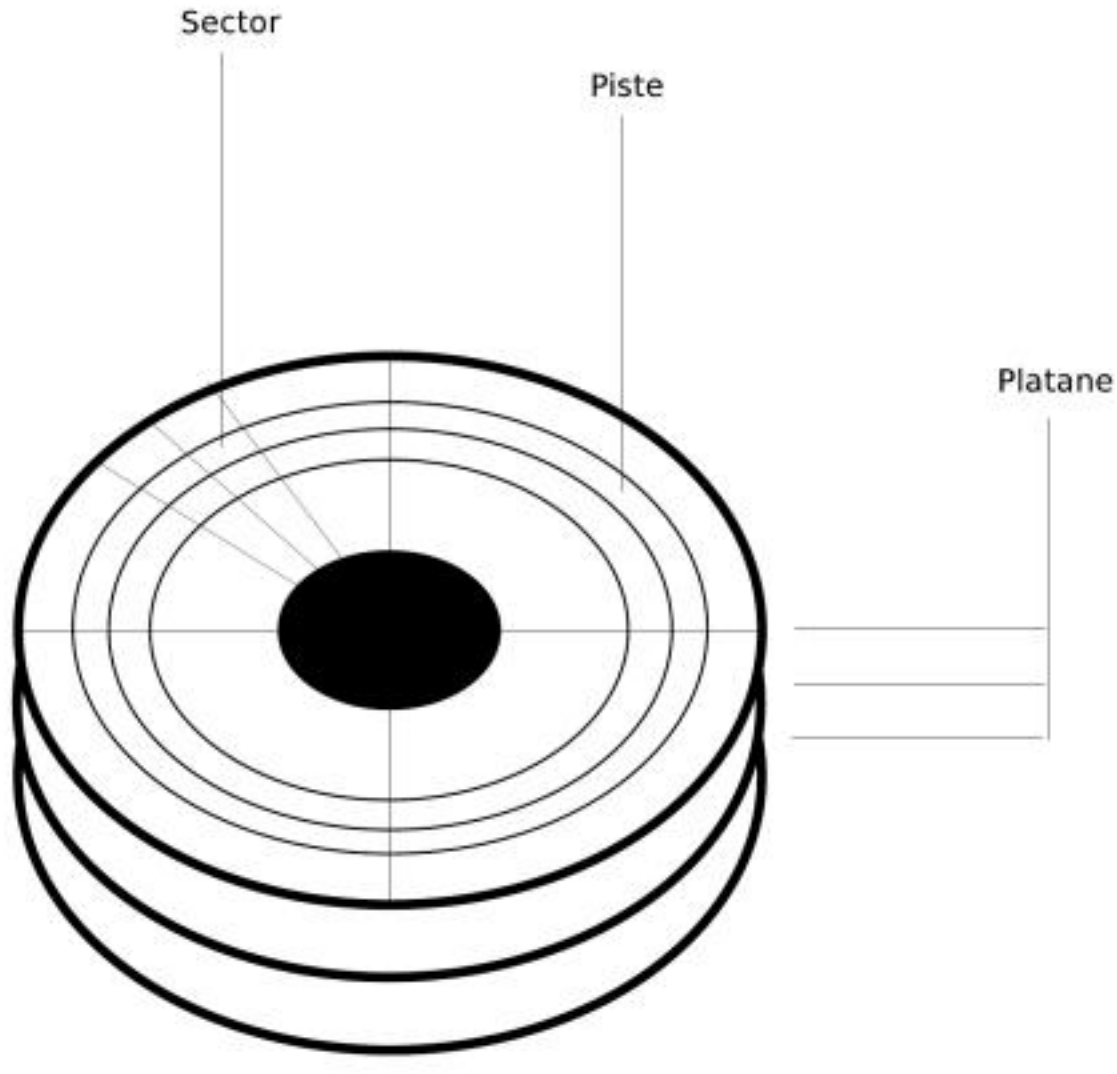


Figura 7.4: Structura unui hard-disk

7.2.4 Placa de bază; interconectarea componentelor

> **Placa de bază** (*mainboard*, *motherboard*) este componenta fizică al cărei rol este asigurarea conectivității celorlalte componente fizice. În mod tipic, o placă de bază are locuri (*sloturi*, *socket-uri*) pentru celelalte componente: procesor, memorie, placă video etc.

După cum se observă și în figura 7.5, placa de bază conține două "nuclee" de interconectare denumite **chipset-uri**. Acestea sunt *northbridge* și *southbridge*. Așa cum rezultă și din denumirea lor, *northbridge*-ul se află în partea de sus a plăcii de bază iar *southbridge*-ul în partea de jos. Există o conexiune a *northbridge* cu *southbridge*-ul.

Northbridge-ul mai este numit și *Memory Controller Hub (MCH)* și este folosit pentru a conecta componentele rapide ale unui sistem de calcul:

- Procesorul
- Memoria RAM
- Placa video (prin PCI Express sau AGP)

Northbridge-ul dictează tipul de procesor, cantitatea, viteza și tipul de memorie RAM care poate fi folosită.

Southbridge-ul mai este numit și *I/O Controller Hub (ICH)* și este folosit pentru interconectarea componentelor mai lente ale sistemului. Conține următoarele componente și magistrale:

- Magistrală PCI pentru legătura cu plăcile PCI
- Magistrală LPC pentru legătura cu Super I/O (tastatură, mouse, port paralel, port serial, controler de floppy)
- Magistrală SPI pentru legătura cu dispozitive de stocare de tip flash
- Magistrală SM pentru legătura cu alte dispozitive de pe placa de baza (senzori de temperatură și ventilator)
- Controler DMA
- Controler de întreruperi
- Dispozitivele de stocare (IDE, SATA)
- Real Time Clock (RTC)
- USB
- Ethernet (Interfața de rețea)
- Interfața audio
- Memorie BIOS

7.2.5 Dispozitive periferice; magistrale

Dispozitivele periferice ale sistemului asigură comunicația acestuia cu exteriorul. Conectarea acestor dispozitive periferice se realizează prin intermediul unor conectori în placa de bază. Acești conectori sunt oferiți fie direct de placa de bază (în cazul unor plăci incorporate), fie de o placă specializată conectată pe placa de bază (placa video, placa de rețea etc.)

Conexiunea diverselor plăci în placa de bază se realizează prin intermediul unor sloturi și a unor magistrale dedicate: PCI, AGP, USB etc.

Magistrale importante ale sistemului sunt:

- FSB – Front Side Bus: conectează procesorul de northbridge
- Back Side Bus – conectează memoria cache de procesor
- PCI – Peripheral Component Interconnection
- AGP – Advanced Graphics Port
- PCI Express
- USB – Universal Serial Bus

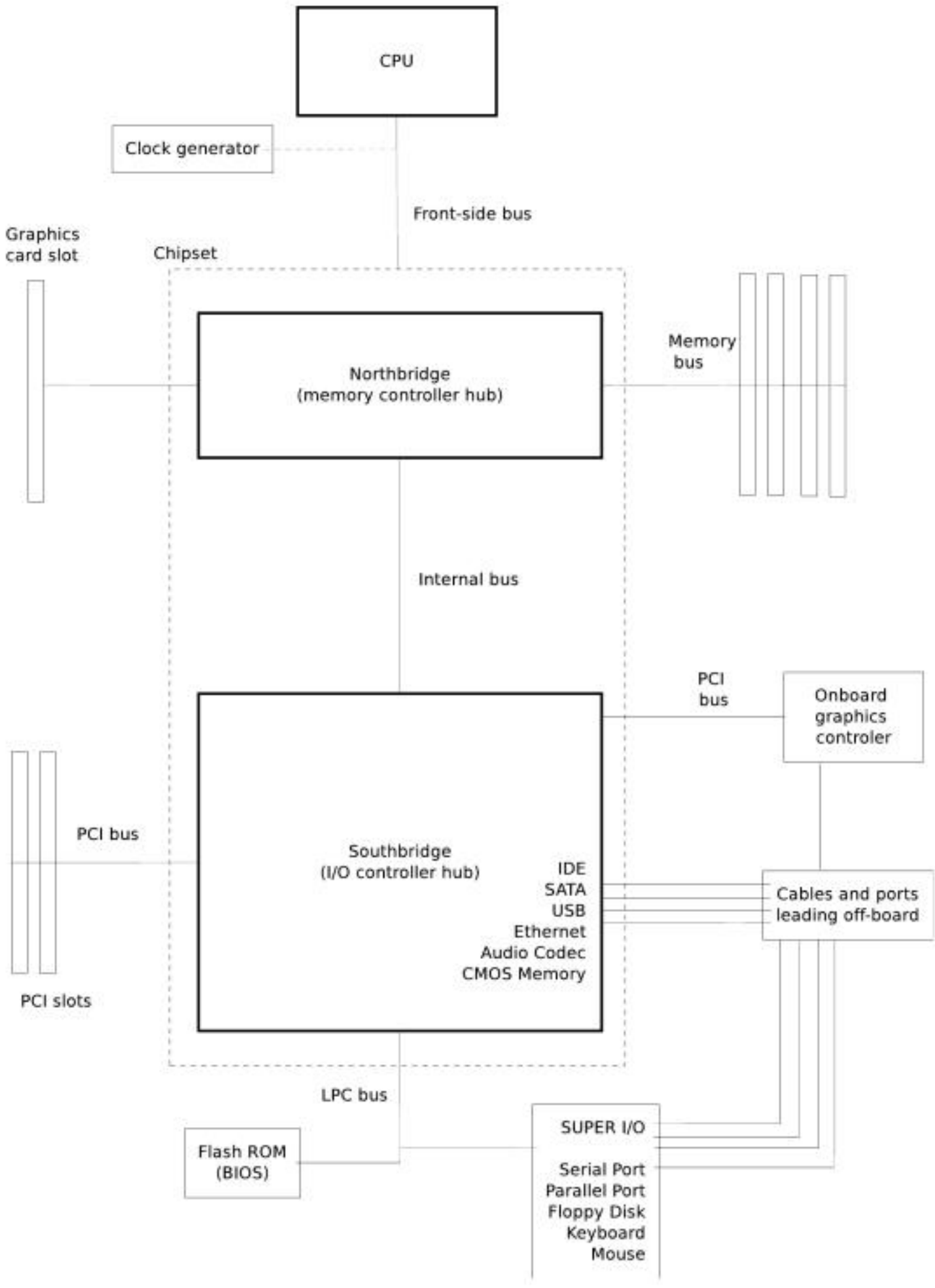


Figura 7.5: Diagrama bloc a plăcii de bază

7.3 Suportul pentru dispozitive la nivelul kernel-ului

Nucleul sistemului de operare este cel care intermediază accesul aplicațiilor la hardware. Pentru ca hardware-ul să poată fi utilizat, nucleul trebuie să știe cum să comunice cu acesta. Nucleul oferă suport de comunicație cu dispozitivele sistemului prin intermediul driver-elor.

Driver-ul reprezintă un set de rutine care se execută la nivelul nucleului și care permit accesul la un anumit dispozitiv.

Dispozitivele pe care le poate utiliza o aplicație pot fi dispozitive fizice reale sau pot fi dispozitive virtuale (care simulează comportamentul unui dispozitiv hardware). În Linux, suportul pentru dispozitive poate fi inclus în kernel (în momentul compilării) sau poate fi disponibil prin intermediul *modulelor*, module care pot fi încărcate la cerere de către sistem.

În prezent, majoritatea distribuțiilor conțin kernel-uri compilate cu suport generic pentru unele dispozitive mai des întâlnite, iar suportul pentru anumite dispozitive este disponibil prin module. Modulele sunt încărcate doar dacă dispozitivele aferente sunt prezente.

7.3.1 Listarea modulelor încărcate la un moment dat în sistem

Pentru a lista modulele încărcate la un moment dat se utilizează comanda **`lsmod`**:

```
ubuntu@ubuntu:~$ lsmod
Module                  Size  Used by
rfcomm                 40856  0
l2cap                  25728  5 rfcomm
bluetooth              55908  4 rfcomm,l2cap
lp                     12452  0
cpufreq_userspace       5408  0
cpufreq_powersave       2688  0
cpufreq_ondemand        9228  0
freq_table              5792  2 cpufreq_stats,cpufreq_ondemand
[...]
```

Comanda afișează pentru fiecare modul în parte, pe coloane:

- numele;
- spațiul de memorie ocupat de acel modul;
- numărul și numele modulele care depind de el.

Același lucru se poate afla prin intermediul sistemului de fișiere virtual `/proc`. Fișierul `/proc/modules` conține lista modulelor încărcate la un moment dat.

```
ubuntu@ubuntu:~$ cat /proc/modules
rfcomm 40856 0 - Live 0xe0bbb000
l2cap 25728 5 rfcomm, Live 0xe0b47000
bluetooth 55908 4 rfcomm,l2cap, Live 0xe0b51000
lp 12452 0 - Live 0xe0b39000
cpufreq_userspace 5408 0 - Live 0xe0b31000
cpufreq_stats 7360 0 - Live 0xe0b2e000
cpufreq_powersave 2688 0 - Live 0xe0859000
cpufreq_ondemand 9228 0 - Live 0xe0aee000
freq_table 5792 2 cpufreq_stats,cpufreq_ondemand, Live 0xe0b27000
[...]
```

Modulele disponibile în sistem se găsesc în `/lib/modules/<versiune_kernel>`. Modulele incluse în nucleul distribuției se găsesc în `/lib/modules/<versiune_kernel>/kernel`. În acest director ele sunt

organizate în funcție de categoria din care fac parte – există astfel directoarele `arch` (module pentru arhitectura curentă), `crypto` (criptografie), `drivers` (driver-e dispozitive), `fs` (sisteme de fișiere), `lib` (module care implementează diferite funcții în kernel), `net` (rețea), `security` (securitate), `sound` (sunet):

```
root@ubuntu:~# ls /lib/modules/2.6.18-4-686/kernel/
arch  crypto  drivers  fs  lib  net  sound
```

Versiunea de kernel poate fi aflată utilizând comanda:

```
ubuntu@ubuntu:~$ uname -r
2.6.20-15-generic
```

După cum s-a observat și din ieșirea comenzii **lsmod**, între module există dependențe. Aceste dependențe sunt descrise în `/lib/modules/<versiune_kernel>/modules.dep` – acest fișier este generat automat la rularea comenzii **depmod**.

7.3.2 Încărcarea unui modul

Înainte de a prezenta modul în care un modul este încărcat în nucleu, trebuie menționat că în distribuțiile moderne Linux modulele se încarcă automat în funcție de dispozitivele hardware prezente în sistem.

Încărcarea manuală a unui modul se poate realiza folosind comanda **insmod** sau comanda **modprobe**. Comanda **insmod** primește ca argument calea către fișierul unde se găsește modulul. Acest lucru o face destul de dificilă de folosit. Un exemplu de utilizare este dat mai jos (se presupune că modulul ce se dorește a fi încărcat nu este deja încărcat).

```
root@ubuntu:~# insmod ./drivers/input/misc/pcspkr.ko

root@ubuntu:~# lsmod | grep pcsp
pcspkr                  4224  0
```

O comandă echivalentă comenzii **insmod** utilizate mai sus dar care încarcă și dependențele unui modul (dacă există) este **modprobe <nume_modul>**:

```
root@ubuntu:~# modprobe pcspkr
```

Pentru a configura **modprobe** se poate utiliza fișierul `/etc/modprobe.conf` sau directorul `/etc/modprobe.d`.

7.3.3 Descărcarea unui modul din kernel

Pentru descărcarea unui modul din kernel se poate utiliza comanda **rmmod** sau comanda **modprobe -r**. Comenzile următoare descarcă modulul `pcspkr`, cu observația că, la utilizarea lui **modprobe**, se încearcă și descărcarea modulelor de care `pcspkr` depinde (dacă există și nu depind și alte module de ele).

```
root@ubuntu:~# rmmod pcspkr

root@ubuntu:~# modprobe -r pcspkr
```

Tabelul 7.1: Ierarhia din /sys

Director	Conținut
`/sys/devices`	Subdirectoarele care reflectă modul fizic de conectare al dispozitivelor în sistem
`/sys/bus`	Legături simbolice către dispozitive grupate în funcție de bus-ul (magistrala) de care aparțin acestea
`/sys/class`	Dispozitivele grupate în funcție de clasa din care fac parte
`/sys/block`	Dispozitivele bloc
`/sys/firmware`	Dispozitivele cu suport firmware
`/sys/fs`	Informații despre sistemele de fișiere
`/sys/kernel`	Informații despre kernel
`/sys/module`	Subdirectoare pentru fiecare modul în parte
`/sys/power`	Informații despre consumul de curent

7.4 Analiza hardware a unui sistem (magistrale, chipset, CPU, memorie, dispozitive)

7.4.1 Lista hardware – /sys

Folosind sistemul de fișiere virtual (`sysfs`) montat de kernel în `/sys`, se pot obține informații despre dispozitivele hardware aflate în system. `/sys` este populat de kernel cu informații provenite de la driver-ele încărcate la un moment dat. Dispozitivele hardware afișate în acest director sunt cele care au fost detectate de kernel și care sunt utilizabile de acesta. Tabelul 7.1 prezintă conținutul directoarelor din `/sys`.

7.4.2 Comenzi pentru afișarea dispozitivelor

Informații despre dispozitivele din sistem și starea acestora pot fi aflate și prin intermediul unor comenzi. În cele ce urmează, se vor prezenta comenzile utilizate pentru a prezenta diverse informații despre dispozitivele din sistem.

Dispozitive USB

Pentru a afișa magistralele și dispozitivele USB prezente în sistem se utilizează comanda **`lsusb`**:

```
1 ubuntu@ubuntu:~$ lsusb
2 Bus 002 Device 001: ID 0000:0000
3 Bus 001 Device 001: ID 0000:0000
```

Informații despre magistrala USB se regăsesc și în `/sys/bus/usb/`. Subdirectorul `devices` prezintă dispozitivele sub formă de legături simbolice către directoare din `/sys/devices`. Subdirectorul `drivers` din `/sys/bus/usb/` conține informații despre driver-ele utilizate.

```
ubuntu@ubuntu:~$ ls -l /sys/bus/usb/devices/
total 0
lrwxrwxrwx 1 root root 0 2007-09-04 18:16 1-0:1.0 ->
../../../devices/pci0000:00/0000:00:07.2/usb1/1-0:1.0
lrwxrwxrwx 1 root root 0 2007-09-04 18:16 2-0:1.0 ->
../../../devices/pci0000:00/0000:00:11.0/0000:02:02.0/usb2/2-0:1.0
lrwxrwxrwx 1 root root 0 2007-09-04 18:16 usb1 ->
../../../devices/pci0000:00/0000:00:07.2/usb1
lrwxrwxrwx 1 root root 0 2007-09-04 18:16 usb2 ->
../../../devices/pci0000:00/0000:00:11.0/0000:02:02.0/usb2

ubuntu@ubuntu:~$ ls -l /sys/bus/usb/drivers/
total 0
drwxr-xr-x 2 root root 0 2007-09-04 18:16 hub
drwxr-xr-x 2 root root 0 2007-09-04 18:16 usb
drwxr-xr-x 2 root root 0 2007-09-04 18:16 usbfs

ubuntu@ubuntu:~$ ls -lR /sys/bus/usb/drivers/ | grep module
lrwxrwxrwx 1 root root    0 module -> ../../../../module/usbcore
lrwxrwxrwx 1 root root    0 module -> ../../../../module/usbcore
lrwxrwxrwx 1 root root    0 module -> ../../../../module/usbcore
```

Ultima comandă listează recursiv directoarele din `/sys/bus/usb/drivers`, căutând liniile care conțin referințe la fișiere cu numele module. Se observă astfel că modulul utilizat este usbcore (datele și orele fișierelor au fost eliminate din ieșirea comenzii).

Dispozitive PCI

Dispozitivele de pe magistrala PCI se pot afla utilizând comanda **`lspci`**:

```
ubuntu@ubuntu:~$ lspci
00:00.0 Host bridge: Intel Corporation 82865G/PE/P DRAM Controller/Host-
Hub
Interface (rev 02)
00:01.0 PCI bridge: Intel Corporation 82865G/PE/P PCI to AGP Controller
(rev 02)
00:1d.0 USB Controller: Intel Corporation 82801EB/ER (ICH5/ICH5R) USB
UHCI
Controller #1 (rev 02)
00:1e.0 PCI bridge: Intel Corporation 82801 PCI Bridge (rev c2)
00:1f.0 ISA bridge: Intel Corporation 82801EB/ER (ICH5/ICH5R) LPC
Interface
Bridge (rev 02)
[...]
```

Similar magistralei USB, se pot determina și directoarele din `/sys` asociate dispozitivelor PCI. Comanda următoare prezintă modul în care se pot afla modulele utilizate de dispozitivele PCI, folosind aceeași procedură ca în cazul dispozitivelor USB (datele și orele fișierelor au fost eliminate din ieșirea comenzii).

```
ubuntu@ubuntu:~$ ls -lR /sys/bus/pci/drivers/ | grep module
lrwxrwxrwx 1 root root    0 module -> ../../../../module/snd_intel8x0
lrwxrwxrwx 1 root root    0 module -> ../../../../module/intel_agp
lrwxrwxrwx 1 root root    0 module -> ../../../../module/ata_generic
lrwxrwxrwx 1 root root    0 module -> ../../../../module/ata_piix
[...]
```

Sistemul ACPI

Sistemele moderne au implementat sistemul ACPI[1] (*Advanced Configuration and Power Interface*) pentru configurarea unor dispozitive și managementul consumului de curent. Fișierele virtuale ale acestui sistem se află în `/proc/acpi`.

Spre exemplu, sistemul ACPI oferă, printre altele, informații despre:

- procesoarele din sistem – `/proc/acpi/processor`
- ventilatoarele din sistem – `/proc/acpi/fan`
- bateriile din sistem – `/proc/acpi/battery`
- senzorii de temperatură – `/proc/acpi/thermal_zone`

hwinfo

`hwinfo` este o aplicație pentru detecția hardware-ului. Comanda **`hwinfo`** afișează în mod implicit toate informațiile despre dispozitivele detectate la un nivel foarte detaliat. Pentru o listare mai succintă se folosește următoarea comandă:

```
1  ubuntu@ubuntu:~$ hwinfo --short
2  cpu:
3                       Intel(R) Pentium(R) Dual  CPU  T2390  @ 1.86GHz,
   800
4  MHz
5                       Intel(R) Pentium(R) Dual  CPU  T2390  @ 1.86GHz,
6  1600 MHz
7  keyboard:
8    /dev/input/event4    AT Translated Set 2 keyboard
9  mouse:
10   /dev/input/mice      Macintosh mouse button emulation
11   /dev/input/mice      SynPS/2 Synaptics TouchPad
12 printer:
13   /dev/usb/lp0         Samsung ML-1640 Series
14 graphics card:
15                      Intel Mobile GM965/GL960 Integrated Graphics
16 Controller
17                      Intel 965 GM
18 sound:
19                      Intel 82801H (ICH8 Family) HD Audio Controller
20 storage:
21                      Intel 82801HBM/HEM (ICH8M/ICH8M-E) SATA IDE
22 Controller
23                      Intel 82801HBM/HEM (ICH8M/ICH8M-E) IDE Controller
24 network:
25   wlan0               Intel PRO/Wireless 3945ABG Network Connection
26   eth0                Realtek RTL8101E PCI Express Fast Ethernet
27 controller
28 network interface:
29   pan0                Ethernet network interface
30   wlan0               WLAN network interface
31   wmaster0            Network Interface
32   eth0                Ethernet network interface
```

[1] ACPI – http://www.acpi.info/

```
  lo                    Loopback network interface
disk:
  /dev/sda              WDC WD2500BEVS-2
[...]
```

Pentru a obține detalii despre o anumită componentă, se folosește numele componentei ca argument al comenzii. De exemplu, pentru procesor se folosește următoarea comandă:

```
ubuntu@ubuntu:~$ hwinfo --cpu
01: None 00.0: 10103 CPU
  [Created at cpu.304]
  Unique ID: rdCR.j8NaKXDZtZ6
  Hardware Class: cpu
  Arch: Intel
  Vendor: "GenuineIntel"
  Model: 6.15.13 "Intel(R) Pentium(R) Dual  CPU  T2390  @ 1.86GHz"
  Features:
fpu,vme,de,pse,tsc,msr,pae,mce,cx8,apic,sep,mtrr,pge,mca,cmov,pat,pse36,
clflush,dts,acpi,mmx,fxsr,sse,sse2,ss,ht,tm,pbe,nx,lm,constant_tsc,
arch_perfmon,pebs,bts,pni,dtes64,monitor,ds_cpl,est,tm2,ssse3,cx16,xtpr,
pdcm,lahf_lm
  Clock: 1866 MHz
  BogoMips: 3724.63
  Cache: 1024 kb
  Units/Processor: 2
  Config Status: cfg=new, avail=yes, need=no, active=unknown
```

7.4.3 Monitorizarea stării dispozitivelor hardware

Senzori

O altă sursă de informații despre hardware-ul prezent într-un sistem o reprezintă senzorii. Acești senzori pot oferi informații despre viteza ventilatoarelor, voltaje și temperatură, fiind folosite pentru a monitoriza starea sistemului.

Pachetul `lm-sensors` include programul **`sensors`** care afișează informații citite de la aceștia.

După instalarea pachetului trebuie rulată comanda **`sensors-detect`** care va detecta senzorii prezenți în sistem și va indica ce module trebuie încărcate. După ce configurarea se realizează cu succes, poate fi utilizată comanda **`sensors`**.

```
ubuntu@ubuntu:~$ sensors
smsc47m192-i2c-0-2d
Adapter: SMBus I801 adapter at 1400
+2.5V:      +2.60 V  (min =  +0.00 V, max =  +3.32 V)
VCore:      +1.42 V  (min =  +0.00 V, max =  +2.99 V)
+3.3V:      +3.13 V  (min =  +0.00 V, max =  +4.38 V)
+5V:        +5.16 V  (min =  +0.00 V, max =  +6.64 V)
[...]
smsc47m1-isa-0800
Adapter: ISA adapter
fan1:      2792 RPM  (min =  640 RPM, div = 8)
fan2:      2118 RPM  (min =  640 RPM, div = 8)
```

Pachetul de utilitare smartmontools

Pachetul `smartmontools` pune la dispoziție programe pentru monitorizarea hard-disk-urilor utilizând tehnologia S.M.A.R.T. (`Self-Monitoring, Analysis and Reporting Technology`). În unele sisteme, trebuie activat suportul S.M.A.R.T. în BIOS (unele BIOS-uri au această configurare în setul de opțiuni avansate). Tehnologia permite monitorizarea indicatorilor de stare a hard-disk-urilor în scopul anticipării posibilelor căderi.

În pachet se găsesc două programe:

- **`smartctl`** – pentru analiza stării S.M.A.R.T. la un moment dat;
- **`smartd`** – un daemon pentru monitorizarea permanentă.

Pentru a verifica starea la un moment dat pentru primul hard-disk, se poate utiliza comanda:

```
root@ubuntu:~# smartctl --all /dev/sda
smartctl version 5.37 [i686-pc-linux-gnu] Copyright (C) 2002-6 Bruce
Allen
Home page is http://smartmontools.sourceforge.net/

=== START OF INFORMATION SECTION ===
Model Family:     Seagate Barracuda 7200.7 and 7200.7 Plus family
Device Model:     ST3120827AS
Serial Number:    4MS03F43
Firmware Version: 3.42
User Capacity:    120,033,041,920 bytes
[...]

SMART Attributes Data Structure revision number: 10
Vendor Specific SMART Attributes with Thresholds:
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED
WHEN_FAILED RAW_VALUE
  1 Raw_Read_Error_Rate     0x000f   062   049   006    Pre-fail  Always
-       16
  3 Spin_Up_Time            0x0003   098   096   000    Pre-fail  Always
-       0
  4 Start_Stop_Count        0x0032   100   100   020    Old_age   Always
-       659
[...]
```

SMART are și suport pentru execuția de teste asupra discului. Există teste lungi și teste scurte. Execuția unui test poate fi urmărită în timp real dar poate fi și programată. Rezultatele testului se pot afla în ieșirea comenzii **`smartctl -all /dev/sda`**.

Un test scurt poate fi programat utilizând comanda:

```
root@ubuntu:~# smartctl --test=short /dev/sda
smartctl version 5.37 [i686-pc-linux-gnu] Copyright (C) 2002-6 Bruce
Allen
Home page is http://smartmontools.sourceforge.net/

=== START OF OFFLINE IMMEDIATE AND SELF-TEST SECTION ===
Sending command: "Execute SMART Short self-test routine immediately in
off-line mode".
Drive command "Execute SMART Short self-test routine immediately in
off-line mode" successful.
```

```
Testing has begun.
Please wait 1 minutes for test to complete.
Test will complete after Thu Sep 13 10:34:15 2007

Use smartctl -X to abort test.
```

Se poate observa la o nouă rulare a comenzii **`smartctl -all /dev/sda`** că apar la ieșire și rezultatele testului (dacă s-a terminat), precum în exemplul de mai jos.

```
root@ubuntu:~# smartctl --all /dev/sda
smartctl version 5.37 [i686-pc-linux-gnu] Copyright (C) 2002-6 Bruce
Allen
Home page is http://smartmontools.sourceforge.net/

=== START OF INFORMATION SECTION ===
[...]

=== START OF READ SMART DATA SECTION ===
SMART overall-health self-assessment test result: PASSED
[...]
```

Pe lângă testul scurt mai există și alte teste (spre exemplu un test lung și un test offline). Mai multe exemple de folosire a lui **`smartctl`** se pot afla din man smartctl și din **`smartctl -h`**.

Daemon-ul **`smartd`** monitorizează hard-disk-urile și permite înregistrarea stării acestora și a defectelor apărute prin intermediul `syslog`. În mod implicit verifică discurile la fiecare 30 minute. Daemon-ul se poate configura prin intermediul fișierului `/etc/smartd.conf`. Pentru lucrul cu daemoni se poate consulta capitolul 5.

7.5 Interfața cu dispozitivele din userspace – `udev` și `/dev`

În Linux există câte un fișier pentru fiecare dispozitiv în directorul `/dev`. Fișierele din `/dev` sunt create de managerul de dispozitive numit **`udev`**. Acesta este executat în întregime ca daemon în spațiul utilizator (user-space) și astfel poate executa programe arbitrare la apariția sau la dispariția dispozitivelor din sistem. **`udev`** oferă suport pentru nume persistente pentru dispozitive, ceea ce înseamnă că poate fi configurat să folosească întotdeauna același nume pentru fișierul asociat unui anumit dispozitiv (în implementările mai vechi pentru `/dev`, la reintroducerea în sistem a unor dispozitive, acestea puteau primi nume diferite comparativ cu introducerea anterioară).

`udev` primește notificări de la kernel atunci când apar sau dispar dispozitive. Astfel, spre exemplu, la conectarea unui USB stick în calculator, kernel-ul notifică **`udev`** că a apărut un nou dispozitiv de stocare. În funcție de schema de nume utilizată, **`udev`** poate, spre exemplu, să aleagă drept nume de dispozitiv următorul nume disponibil după numele hard-disk-ul prezent în sistem (presupunând că dispozitivul asociat hard-disk-ului este `/dev/sda`, udev alege ca nume `/dev/sdb`). După ce este ales un nume de dispozitiv, se poate executa un program care, în cazul prezentat aici, să monteze sistemul de fișiere de pe USB stick într-un subdirector din `/media`. Tot în funcție de configurări, anumite dispozitive pot să aibă fișierele asociate puse într-un subdirector separat din `/dev`.

Tabelul 7.2: Convenția de nume pentru dispozitive în Linux

Nume	Dispozitiv
`fd`	Floppy-disk
`hd`	Hard-disk IDE
`lp`	Imprimantă
`sd`	Hard-disk SATA (SCSI)
`tty`	Terminal fizic
`ram`	Dispozitiv care reprezintă memoria RAM
`loop`	Dispozitiv utilizat pentru montarea fișierelor precum dispozitivele reale

Pe lângă execuția de programe, **udev** mai permite modificarea permisiunilor pentru fișierele din `/dev` create. Astfel se poate configura, spre exemplu, ca pentru orice scanner introdus în sistem, fișierul creat să aparțină grupului cu numele scanner. În acest fel, numai utilizatorii care fac parte din acest grup vor putea lucra cu acest fișier și implicit cu scanner-ul. Trebuie amintit faptul că un utilizator poate să fie membru al mai multor grupuri.

Configurările pentru **udev** se găsesc în `/etc/udev/rules.d`. Acest director conține fișiere care descriu modul în care, în funcție de dispozitiv, se formează numele dispozitivelor, se dau permisiuni fișierelor sau se execută programe. De obicei acest director conține configurări aplicabile în orice situație și nu se recomandă editarea fișierelor din el.

7.6 Lucrul cu dispozitive (`/dev`)

S-a menționat anterior că **udev** se ocupă și de stabilirea permisiunilor pentru anumite dispozitive. Un utilizator trebuie să se găsească în grupul din care face fișierul dispozitivului pentru a putea utiliza dispozitivul (prin intermediul fișierului, bineînțeles).

Tabelul 7.2 prezintă câteva nume asociate în mod normal cu dispozitivele într-un mediu Linux. Trebuie menționat că de obicei numele sunt urmate de un număr sau o literă, care indică al câtelea dispozitiv de acel tip este în sistem. Spre exemplu, două unități de dischetă vor fi numite `/dev/fd0` și `/dev/fd1`, iar două hard-disk-uri vor fi numite `/dev/sda` și `/dev/sdb`.

Comanda următoare prezintă conținutul tipic pentru `/dev`:

```
1  ubuntu@ubuntu:~$ ls -l /dev/
2  total 0
3  lrwxrwxrwx 1 root root 10 2007-08-31 14:48 adsp -> sound/adsp
4  lrwxrwxrwx 1 root root 12 2007-08-31 14:48 agpgart -> misc/agpgart
5  lrwxrwxrwx 1 root root 11 2007-08-31 14:48 audio -> sound/audio
6  drwxr-xr-x 3 root root  0 2007-08-31 14:48 bus
7  drwxr-xr-x 2 root root  0 2007-08-31 14:48 cd
8  lrwxrwxrwx 1 root root 17 2007-08-31 14:48 cdrom -> /dev/cd/cdrom-sr0
9  lrwxrwxrwx 1 root root 17 2007-08-31 14:48 cdrom0-> /dev/cd/cdrom-sr0
10 [...]
11 drwxr-xr-x 2 root root  0 2007-08-31 14:48 loop
12 lrwxrwxrwx 1 root root  6 2007-08-31 14:48 loop0 -> loop/0
13 lrwxrwxrwx 1 root root  6 2007-08-31 14:48 loop1 -> loop/1
```

```
14 crw-r----- 1 root kmem  1,    1 2007-08-31 14:48 mem
15 lrwxrwxrwx 1 root root 11 2007-08-31 14:48 mixer -> sound/mixer
16 crw-rw-rw- 1 root root  1,    3 2007-08-31 14:48 null
17 [...]
18 crw-rw-rw- 1 root root  1,    8 2007-08-31 14:48 random
19 brw-rw---- 1 root disk  8,    0 2007-08-31 14:48 sda
20 brw-rw---- 1 root disk  8,    1 2007-08-31 14:48 sda1
21 brw-rw---- 1 root disk  8,    2 2007-08-31 14:48 sda2
22 brw-rw---- 1 root disk  8,   16 2007-08-31 14:48 sdb
23 [...]
```

7.6.1 Tipuri de dispozitive

În Linux, dispozitivele se împart în 2 categorii, fiecare categorie având particularitățile ei:

- dispozitive caracter
- dispozitive bloc

Dispozitivele pot fi diferențiate după litera care apare pe prima coloană în ieșirea comenzii **`ls -l`**, coloană care indică tipul fișierului. În lista de mai sus fișierele care au tipul `c` indică dispozitive de tip caracter iar fișierele cu tipul `b` indică dispozitive de tip bloc.

Câteva dispozitive sunt generate de kernel și au un comportament aparte. Aceste dispozitive sunt numite pseudo-dispozitive.

Dispozitivele caracter

> **Dispozitivele caracter** corespund dispozitivelor care transmit date câte un caracter odată.

Aceste dispozitive sunt de obicei folosite pentru a transmite fluxuri de date, precum:

- terminalul – `/dev/tty0`, `/dev/tty1`
- porturile seriale – `/dev/ttyS0`, `/dev/ttyS1`
- dispozitivele audio – `/dev/sequencer`, `/dev/mixer`, `/dev/midi`

Dispozitivele bloc

> Dispozitivele bloc corespund dispozitivelor care lucrează cu datele la nivel de bloc (scriu, citesc, șterg blocuri de date).

Dispozitivele care funcționează în acest fel sunt dispozitivele de stocare:

- fizice (hard disk-uri, unități CD-ROM/DVD-ROM etc.) – `/dev/sda`, `/dev/sda1`, `/dev/hda`
- virtuale (dispozitivele loop) – `/dev/loop0`

- asociate zonelor de memorie – `/dev/ram0`

Pseudo-dispozitive

Există o serie de dispozitive prezente în `/dev` care nu corespund unor componente hardware. Acestea sunt echivalente unor dispozitive virtuale care implementează comportamentul dispozitivelor normale de tip caracter.

Cele mai cunoscute sunt:

- `/dev/null` – fișier în care se poate scrie orice și totul este pierdut – de obicei ieșirile standard sau de eroare ale programelor sunt redirectate către `/dev/null` pentru a suprima afișarea oricărui mesaj;
- `/dev/zero` – fișier din care se poate citi la infinit octetul cu valoarea 0 – de obicei este folosit pentru a suprascrie zone de memorie/disc cu zerouri sau pentru a umple fișiere;
- `/dev/urandom` – fișier din care se pot citi la infinit octeți cu valori aleatoare – utilizat de obicei pentru a scrie fișiere sau zone de pe disc cu date aleatoare.

Câteva exemple de utilizare sunt prezentate mai jos.

Suprimarea ieșirii standard pentru un program:

```
1 ubuntu@ubuntu:~$ ./program > /dev/null
```

Suprimarea ieșirii de eroare pentru un program (mesajele de eroare se pierd):

```
1 ubuntu@ubuntu:~$ ./program 2>/dev/null
```

Suprimarea ambelor ieșiri (standard și eroare) pentru un program:

```
1 ubuntu@ubuntu:~$ ./program 2>&1 >/dev/null
```

Suprascrierea primei partiții de pe primul hard-disk cu zerouri:

```
1 root@ubuntu:~# dd if=/dev/zero of=/dev/sda1
```

Suprascrierea primului hard-disk cu date aleatoare:

```
1 root@ubuntu:~# dd if=/dev/urandom of=/dev/sda
```

7.6.2 Întreruperi hardware

> O întrerupere (**IRQ** – *interrupt request*) este un semnal trimis de un dispozitiv către procesor pentru a-l anunța că trebuie să-și oprească activitatea pentru a procesa un eveniment extern, cum ar fi apăsarea unei taste.

Pe platforma x86 sunt 16 IRQ-uri numerotate de la 0 la 15. Pe calculatoarele moderne pot exista mai mult de 16 întreruperi. Unele întreruperi sunt rezervate pentru dispozitive specifice, cum ar fi tastatura, iar altele pot fi reasignate. În tabelul 7.3 se află întreruperile și semnificația lor pentru platforma x86:

Tabelul 7.3: Semnificația întreruperilor

IRQ	Semnificație
0	System timer
1	Tastatura
2	Cascada pentru IRQ 8-15
3	Al doilea port serial (COM 2)
4	Primul port serial (COM 1)
5	Placa de sunet sau al doilea port paralel (LPT2)
6	Controler pentru Floppy
7	Primul port paralel (LPT1)
8	Real-time clock
9	Nerezervat
10	Nerezervat
11	Nerezervat
12	Mouse PS/2
13	Coprocesor aritmetic
14	Controler ATA primar
15	Controler ATA secundar

În cadrul sistemului de operare Linux, semnificația întreruperilor poate fi vizualizată folosind urmatoarea comandă:

```
ubuntu@ubuntu:~$  cat /proc/interrupts
           CPU0
  0:        130   IO-APIC-edge      timer
  1:        277   IO-APIC-edge      i8042
  6:          5   IO-APIC-edge      floppy
  7:          0   IO-APIC-edge      parport0
  8:          3   IO-APIC-edge      rtc
  9:          0   IO-APIC-fasteoi   acpi
 12:       1485   IO-APIC-edge      i8042
 14:          0   IO-APIC-edge      libata
 15:      13885   IO-APIC-edge      libata
 16:       4426   IO-APIC-fasteoi   eth1
 17:      43084   IO-APIC-fasteoi   ioc0
NMI:          0
LOC:     254957
ERR:          0
MIS:          0
```

În comanda anterioară, pentru fiecare întrerupere este afișat driver-ul care o folosește.

7.6.3 Adrese I/O

Adresele I/O, numite și porturi I/O, sunt locații unice în memorie rezervate pentru comunicarea între procesor și diversele dispozitive fizice. În tabelul 7.4 se poate observa corespondența între denumirea dispozitivului pe Linux și pe Windows și adresa I/O asociată:

În Linux se pot verifica adresele I/O folosind următoarea comandă:

Tabelul 7.4: Adresele I/O ale dispozitivelor

Denumire Linux	Denumire Windows	IRQ asociat	Adresa I/O
/dev/ttyS0	COM1	4	0x03f8
/dev/ttyS1	COM2	3	0x02f8
/dev/ttyS2	COM3	4	0x03e8
/dev/ttyS3	COM4	3	0x02e8
/dev/lp0	LPT1	7	0x0378-0x037f
/dev/lp1	LPT2	5	0x0278-0x027f

```
ubuntu@ubuntu:~$ cat /proc/ioports
0000-001f : dma1
0020-0021 : pic1
0040-0043 : timer0
0050-0053 : timer1
0060-006f : keyboard
0070-0077 : rtc
0080-008f : dma page reg
00a0-00a1 : pic2
00c0-00df : dma2
00f0-00ff : fpu
```

7.6.4 Adrese DMA

DMA (*Direct Memory Access*) se referă la adresarea directă a memoriei și este o alternativă la adresele I/O. DMA face posibil transferul datelor între dispozitive și memorie fără ajutorul procesorului. Acest lucru duce la îmbunătățirea performanțelor sistemului deoarece procesorul nu mai este ocupat cu operații I/O.

Arhitectura x86 implementează acest concept folosind canale DMA, câte unul pentru fiecare dispozitiv. Următoarea comandă va afișa ce canale DMA sunt folosite de sistem:

```
ubuntu@ubuntu:~$ cat /proc/dma
 2: floppy
 4: cascade
```

7.6.5 Manipularea datelor la nivel scăzut (comanda dd)

Pentru a lucra cu datele la nivel de octet sistemele de operare de tip Unix pun la dispoziție comanda **dd**. Această comandă poate fi utilizată pentru a manevra date de orice dimensiune. Se bazează pe faptul că datele sunt transferate de la o sursă la o destinație, eventual cu simple transformări.

Comanda primește sursa și destinația sub forma a doi parametri:

- **sursa**: `if=<fișier_intrare>` (dacă lipsește acest parametru se presupune că sursa este intrarea standard)
- **destinația**: `of=<fișier_ieșire>` (dacă lipsește acest parametru se presupune că destinația este ieșirea standard)

Fișierele pot fi atât fișiere normale cât și fișiere care reprezintă dispozitive bloc sau dispozitive caracter. Astfel, o utilizare foarte simplă a comenzii **dd** este următoarea:

```
ubuntu@ubuntu:~$ dd if=/dev/zero of="myfile"
```

Comanda de mai sus copiază de la intrare la ieșire octeți până când apare o eroare, din moment ce nu a fost specificată nicio limită. În aceste condiții, comanda se termină doar dacă fișierul sursă e limitat sau fișierul destinație e limitat. În cazul de mai sus, sursa are dimensiune infinită iar destinația e limitată doar de dimensiunea spațiului liber disponibil pe partiția unde se scrie fișierul.

Pentru a specifica dimensiunea care se dorește a fi copiată se utilizează o combinație a următorilor parametri:

- **numărul de blocuri**: `count=<număr_blocuri>` – dacă acest parametru lipsește copierea se oprește doar în condițiile prezentate mai sus
- **dimensiunea unui bloc**: `bs=<dimensiune_în_octeți>` (*block size*) – pot fi folosiți și multiplii, spre exemplu: `bs=1048576`, `bs=1024K`, `bs=1M` sunt toate echivalente; dacă lipsește acest parametru se utilizează dimensiunea implicită a unui bloc de 512 octeți (dimensiunile blocurilor citite și dimensiunile blocurilor scrise pot fi ajustate individual prin utilizarea parametrilor `ibs` și `obs`; ele sunt modificate simultan prin utilizarea parametrului `bs`)

Exemplele următoare ilustrează utilizarea comenzii dd împreună cu parametrii precizați anterior:

- copierea de 10 MB de date din 10 blocuri de 1 MB (1024K); copierea se face din `/dev/zero` către `/dev/null` deci nu are niciun efect:

```
ubuntu@ubuntu:~$ dd if=/dev/zero of=/dev/null bs=1024K count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 0.00490124 seconds, 2.1 GB/s
```

- același efect ca mai sus doar că se precizează M în loc de K:

```
ubuntu@ubuntu:~$ dd if=/dev/zero of=/dev/null bs=1M count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 0.00509437 seconds, 2.1 GB/s
```

- copierea de 10MB de date în blocuri de 1M la intrare și 128K la ieșire:

```
ubuntu@ubuntu:~$ dd if=/dev/zero of=/dev/null count=10 ibs=1M obs
=128K
10+0 records in
80+0 records out
10485760 bytes (10 MB) copied, 0.00622419 seconds, 1.7 GB/s
```

Comanda **dd** permite și poziționarea în fișierele utilizate pentru intrare și ieșire:

- intrare: `skip=<număr_blocuri>` – sare `<număr_blocuri>` din fișierul de intrare, echivalentul a `<număr_blocuri> * ibs` octeți;
- ieșire: `seek=<număr_blocuri>` – sare `<număr_blocuri>` din fișierul de ieșire, echivalentul a `<număr_blocuri> * obs` octeți.

Tabelul 7.5: Informații expuse prin intermediul lui /proc

Fișier	Conținut
`/proc/cmdline`	Linia care este folosită la inițializarea kernel-ului, inclusiv parametrii trimiși acestuia
`/proc/cpuinfo`	Informații despre procesorul/procesoarele din sistem
`/proc/devices`	Lista dispozitivelor caracter și a celor bloc
`/proc/interrupts`	Statistici despre numărul de întreruperi și dispozitivele asociate întreruperilor
`/proc/meminfo`	Informații despre utilizarea memoriei
`/proc/modules`	Lista modulelor încărcate la un moment dat în sistem – comanda **`lsmod`** prezintă conținutul fișierului într-un mod user-friendly
`/proc/partitions`	Lista partițiilor
`/proc/swaps`	Lista dispozitivelor pe care se face swap
`/proc/uptime`	Timpul trecut de la ultima pornire a sistemului, în secunde – comanda **`uptime`** prezintă conținutul fișierului într-un mod user-friendly
`/proc/version`	Versiunea kernel-ului care rulează la un moment dat

Poziționarea este utilă în cazul în care se dorește ca de pe dispozitivul de intrare să se citească o anumită zonă, precum un anumit sector de pe hard disk. Exemplul următor ilustrează acest lucru – se sare peste primul sector și se citesc două sectoare de pe hard disk (trebuie menționat că dimensiunea unui sector pe HDD este 512 octeți, valoarea implicită pentru parametrul `bs`):

```
1 root@ubuntu:~# dd if=/dev/sda of=/dev/null skip=1 count=2
2 2+0 records in
3 2+0 records out
4 1024 bytes (1.0 kB) copied, 0.043731 seconds, 23.4 kB/s
```

7.7 Analiza sistemului

7.7.1 Informații despre sistem

Prin intermediul directorului `/proc`, kernel-ul expune și o serie de informații legate de sistem. Tabelul 7.5 următor prezintă fișierele mai importante din `/proc` și conținutul acestora.

Conținutul pentru oricare din fișierele prezentate mai sus se poate afla folosind comanda **`cat`**. De exemplu, folosind comanda **`cat`** cu argumentul `/proc/cpuinfo` se vor afișa următoarele informații:

```
1 ubuntu@ubuntu:~$ cat /proc/cpuinfo
2 processor       : 0
3 vendor_id       : GenuineIntel
4 cpu family      : 6
5 model           : 15
6 model name      : Intel(R) Core(TM)2 Duo CPU     T7500  @ 2.20GHz
7 stepping        : 8
```

```
cpu MHz         : 2194.893
cache size      : 4096 KB
fdiv_bug        : no
[...]
```

Folosind comanda anterioară s-au obținut următoarele informații: `processor` reprezintă identificatorul procesorului prezentat (pe sistemele cu un singur procesor, va apărea doar procesorul cu identificatorul 0), `cpu family` reprezintă un număr care se va apenda la numărul 86 pentru a obține numele familiei de procesoare (în exemplul de față 686), `model name` reprezintă denumirea procesorului, `cpu MHz` reprezintă frecvența procesorului iar `cache size` este dimensiunea memoriei cache L2.

Pentru a obține informații despre memoria utilizată de sistem, se va afișa conținutul fișierului `/proc/meminfo`:

```
ubuntu@ubuntu:~$ cat /proc/meminfo
MemTotal:       255676 kB
MemFree:          9684 kB
Buffers:         10668 kB
Cached:          84448 kB
SwapCached:       3004 kB
Active:         179812 kB
Inactive:        43268 kB
SwapTotal:      401400 kB
SwapFree:       360748 kB
[...]
```

Folosind comanda anterioară s-au afișat următoarele informații: `MemTotal` este cantitatea totală de RAM, `MemFree` este cantitatea de RAM care nu este utilizată de sistem, `Buffers` este cantitatea de RAM folosită de buffere, `Cached` este cantitatea de RAM folosită ca memorie cache, `SwapCached` este cantitatea de memorie swap folosită ca memorie cache, `Active` este cantitatea de buffere sau memorie cache care este în curs de utilizare, `Inactive` este cantitatea de buffere sau memorie cache care este liberă, `SwapTotal` este cantitatea totală de swap disponibilă și `SwapFree` este cantitatea de swap liberă.

Folosind următoarea comandă se vor afișa: versiunea kernelului Linux, a compilatorului **gcc** și a distribuției de Linux folosite:

```
ubuntu@ubuntu:~$ cat /proc/version
Linux version 2.6.24-24-generic (buildd@rothera) (gcc version 4.2.4 (
Ubuntu
4.2.4-1ubuntu3)) #1 SMP Tue Aug 18 17:04:53 UTC 2009
```

Oricare dintre aceste fișiere poate fi folosit pentru a găsi, la un moment dat, o informație despre sistem. Spre exemplu, dacă se dorește aflarea tipului procesorului/procesoarelor din sistem se poate utiliza comanda următoare:

```
ubuntu@ubuntu:~$ cat /proc/cpuinfo | grep "model name"
model name      : Intel(R) Core(TM)2 Duo CPU     T7500  @ 2.20GHz
```

Este important de reținut că `/proc` expune informații care sunt accesibile și prin intermediul unor comenzi. Aceste informații, fie că sunt citite direct din `/proc`, fie că sunt citite prin intermediul comenzilor, pot fi filtrate pentru a se extrage date utile pentru utilizator.

7.7.2 Kernel "tunables"

`sysctl` este o interfață care permite configurarea unor parametri din kernel în timpul execuției acestuia. Parametrii pot fi modificați în două feluri:

- prin modificarea directă a fișierelor din `/proc/sys`;
- prin utilizarea comenzii **`sysctl`**.

Primul caz presupune scrierea și citirea parametrilor din anumite căi, precum în exemplul următor:

```
root@ubuntu:~# cat /proc/sys/net/ipv4/ip_forward
0

root@ubuntu:~# echo "1" > /proc/sys/net/ipv4/ip_forward

root@ubuntu:~# cat /proc/sys/net/ipv4/ip_forward
1
```

Comenzile de mai sus au activat rutarea pachetelor de rețea la nivelul kernel-ului, scriind "1" în `/proc/sys/net/ipv4/ip_forward`. Această facilitate este utilă dacă se dorește ca un sistem care are acces la Internet să permită accesul unei întregi subrețele.

Folosind comanda **`sysctl`**, același lucru se obține în felul următor:

```
root@ubuntu:~# sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 0

root@ubuntu:~# sysctl -w net.ipv4.ip_forward="1"
net.ipv4.ip_forward = 1
```

Se observă că pentru a afișa starea unui parametru se folosește o adresare asemănătoare cu cea a structurilor din C/C++. Comanda a doua modifică un parametru prin utilizarea opțiunii `--w` (write).

Modificările prezentate anterior sunt modificări temporare, ele fiind pierdute în cazul în care se repornește sistemul. Pentru a efectua configurări permanente, acestea se scriu în fișierul `/etc/sysctl.conf`. Presupunând că se dorește dezactivarea forwarding-ului, sintaxa în fișier este destul de simplă (liniile care încep cu `#` din fișier sunt comentarii):

```
ubuntu@ubuntu:~$ cat /etc/sysctl.conf
[...]
# Controls IP packet forwarding
net.ipv4.ip_forward = 1
[...]
```

Fișierul de configurare este citit la fiecare pornire a sistemului. Pentru a forța citirea fișierului de configurare, se poate utiliza comanda **`sysctl -p`**:

```
root@ubuntu:~# sysctl -p
net.ipv4.ip_forward = 1
```

7.8 Studii de caz

7.8.1 Salvarea și restaurarea MBR și a tabelei extinse de partiții

MBR-ul (*Master Boot Record*) reține informații despre partițiile primare aflate pe un hard-disk precum atribute ale acestora, poziția lor pe disc și dimensiunea lor. MBR-ul mai conține și bootloader-ul – un mic program care pornește un sistem de operare prezent pe computer. Într-un mediu mixt Windows/Linux, la o instalare a Windows-ului se va observa că Linux-ul devine inaccesibil (bootloader-ul Windows-ului nu cunoaște nimic despre Linux). În aceste condiții, pentru a preîntâmpina acest lucru, se recomandă realizarea unui back-up pentru MBR. Mai multe detalii despre MBR găsiți în secțiunea 6.2.2.

Backup-ul MBR-ului de pe primul HDD se realizează cu următoarea comandă:

```
root@ubuntu:~# dd if=/dev/sda of=mbr.img bs=512 count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 4.8004e-05 seconds, 10.7 MB/s
```

În urma execuției comenzii se realizează fișierul `mbr.img` de 512 octeți cu MBR-ul. Acest fișier poate fi plasat într-un loc sigur și accesibil în cazul în care discul devine inaccesibil (pe un USB stick sau CD-ROM spre exemplu).

Pentru a restaura MBR-ul, se efectuează operația inversă:

```
root@ubuntu:~# dd if=mbr.img of=/dev/null bs=512 count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 4.622e-05 seconds, 11.1 MB/s
```

În MBR se poate stabili ca una dintre partiții să fie partiție extinsă. În partiția extinsă pot fi create mai multe partiții logice, pentru o mai bună organizare a discului. Informațiile despre partițiile logice nu sunt ținute în MBR, deoarece nu încap în cei 512 octeți. Partițiile logice sunt reținute sub formă de listă înlănțuită în afara MBR-ului.

În momentul în care se lucrează cu schema de partiții de pe un hard disk, se recomandă realizarea unui backup înainte. Modificarea schemei de partiții nu șterge datele de pe hard disk, doar schimbă descrierea partițiilor (locul unde se găsesc acestea). În situația în care se șterge o partiție importantă (fie ea logică sau primară), este util să existe un backup al acestor informații pentru recuperare. Modul de backup pentru MBR a fost descris anterior.

Pentru a realiza backup-ul configurației tuturor partițiilor se utilizează programul **`sfdisk`**. Acesta analizează atât partițiile primare cât și pe cele logice, dar nu se uită la informațiile din MBR care nu sunt legate de partiționare (precum bootloader). El este un bun complement pe lângă metoda amintită mai sus de a face backup la MBR.

Comanda următoare face backup pentru întreaga schemă de partiționare folosind **`sfdisk`** – fișierul de backup este `/usbstick/partitions.backup`:

```
root@ubuntu:~# sfdisk -d /dev/sda > /usbstick/partitions.backup
```

Pentru a restaura schema de partiționare salvată anterior, **`sfdisk`** se utilizează astfel:

```
root@ubuntu:~# sfdisk /dev/sda < /usbstick/partitions.backup
```

7.8.2 Salvarea conținutului unui disc cu sectoare inaccesibile

Există situații când, de obicei după o perioada mai lungă de utilizare, un hard disk începe să nu mai funcționeze corect. Se observă acest lucru din erorile pe care sistemul de operare le afișează când vrea să citească date de pe disc. În asemenea situații, cea mai rapidă metodă de backup este copierea fișierelor pe un alt disc.

Acesta este cazul favorabil, când datele mai pot fi identificate și sistemul de operare mai funcționează. În cazul în care defectul hard disk-ului apare în zona în care este descris sistemul de fișiere, există o probabilitate destul de mare ca datele să nu mai poată fi salvate prin simpla copiere a fișierelor. În această situație se poate opta pentru repararea sistemului de fișiere utilizând aplicația **`fsck`**, dar nu se dorește ca această reparare să aibă loc pe hard disk-ul defect. Întâi se realizează o copie a tuturor datelor de pe primul hard disk pe un alt hard disk, copie numită imagine.

Comanda utilizată pentru a realiza această imagine este dd, împreună cu parametrul `conv=noerror`, parametru care îi spune dd-ului să nu se oprească atunci când detectează un bloc inaccesibil.

```
root@ubuntu:~# dd if=/dev/sda of=/dev/sdb conv=noerror
```

Execuția acestei comenzi poate dura foarte mult timp, datorită numărului mare de blocuri care trebuie citite și scrise. Pentru a accelera execuția comenzii, se poate mări dimensiunea blocului citit/scris prin intermediul parametrului `bs` (block size), prezentat anterior. Utilizarea unui bloc mai mare decât dimensiunea unui sector de hard disc (512 octeți) poate duce la pierderea ultimelor sectoare de pe discul sursă, căci nu se poate realiza un bloc cu ele.

7.8.3 Crearea unei imagini de CD; montarea unei imagini

O imagine de CD se creează în mod asemănător creării unei imagini de hard disk.

```
root@ubuntu:~# dd if=/dev/hda of=cd.iso
```

Această comandă va citi CD-ul introdus în unitatea `/dev/hda` sector cu sector și va scrie imaginea lui în fișierul `cd.iso`.

Pentru a putea utiliza imaginea, aceasta trebuie montată într-un director precum orice alt sistem de fișiere, dar cu următoarele observații:

- tipul sistemului de fișiere de pe un CD este `iso9660`;
- deoarece fișierul nu este un dispozitiv real, trebuie utilizată și opțiunea `loop`.

```
root@ubuntu:~# mount -t iso9660 -o loop cd.iso /mnt/cd
```

Pentru a demonta imaginea, se utilizează comanda:

```
root@ubuntu:~# umount /mnt/cd
```

În acest fel pot fi păstrate pe disc imagini ale CD-urilor utilizate des și/sau pot fi testate imagini de CD create de diferite utilitare.

Trebuie menționat că se poate aplica aceeași procedură și pentru DVD-uri, atâta timp cât sistemul de fișiere este corect precizat în comanda **mount** (DVD-urile pot fi scrise în formatul UDF).

7.8.4 Utilizarea unui fișier de pe o partiție FAT32 ca fișier de swap pentru un LiveCD Linux

Se observă de multe ori că LiveCD-urile funcționează greu pe sisteme care nu au multă memorie RAM disponibilă. Într-un sistem instalat, memoria RAM poate fi ajutată și de prezența spațiului de swap de pe hard disk. Într-un sistem în care nu există nicio partiție de swap, un LiveCD nu activează niciun astfel de spațiu. O posibilitate o reprezintă utilizarea unui fișier de swap, pe o partiție deja existentă în sistem. S-a utilizat pentru exemplificare o partiție FAT32, dar se poate utiliza orice tip de partiție pe care Linux-ul poate scrie.

Pentru început, se montează sistemul de fișiere (partiția a doua, primul hard disk) unde se dorește crearea fișierului de swap:

```
1 root@ubuntu:~# mount -t vfat /dev/sda2 /mnt/temp
```

În acest moment se poate crea un fișier de swap de dimensiunea dorită utilizând programul **dd**:

```
1 root@ubuntu:~# dd if=/dev/zero of=/mnt/temp/fisier.swap bs=1M count=256
2 256+0 records in
3 256+0 records out
4 268435456 bytes (268 MB) copied, 6.41171 seconds, 41.9 MB/s
```

S-a creat un fișier de 256 MB (plin cu zerouri) pentru a fi folosit ca fișier de swap. În acest moment trebuie transformat fișierul în zonă de swap, printr-un proces asemănător formatării partițiilor:

```
1 root@ubuntu:~# mkswap /mnt/temp/fisier.swap
```

În acest moment, fișierul este pregătit pentru a fi folosit ca zonă de swap. Sistemul de operare trebuie să afle de prezența fișierului de swap printr-o operație asemănătoare montării unui sistem de fișiere:

```
1 root@ubuntu:~# swapon /mnt/temp/fisier.swap
```

După execuția acestei comenzi sistemul folosește fișierul creat mai sus ca zonă de swap.

Cuvinte cheie

- kernel
- kernel-space
- user-space
- mașină virtuală
- von Neumann
- CPU, procesor

- RAM, DRAM
- placă de bază
- chipset
- northbridge
- southbridge
- hard-disk, HDD
- magistrală, PCI, AGP
- dispozitiv
- modul
- driver
- lsmod, insmod, modprobe, rmmod
- /sys
- lsusb, lspci
- sensors
- smartmontools
- /dev
- udev
- dispozitive de tip bloc
- dispozitive de tip caracter
- pseudodispozitive
- dd
- /proc
- sysctl
- /etc/sysctl.conf
- MBR

Întrebări

1. Care dintre următoarele NU este o componentă a arhitecturii von Neumann?
   - ❑ CPU
   - ❑ unitatea de comandă
   - ❑ memoria
   - ❑ dispozitivele de intrare ieșire
2. Care utilitar poate fi folosit pentru efectuarea unui backup la MBR?
   - ❑ **`lspci`**
   - ❑ **`lsusb`**
   - ❑ **`sysctl`**
   - ❑ **`dd`**
3. Care din următoarele NU oferă direct informații despre sistem?
   - ❑ `/sys`
   - ❑ `/proc`
   - ❑ **`dd`**
   - ❑ **`lsusb`**
4. Care comandă NU este folosită pentru interacțiunea cu modulele din kernel?
   - ❑ **`lsmod`**
   - ❑ **`modprobe`**

- ❑ **modinstall**
- ❑ **insmod**

5. Care din următoarele NU este un exemplu de pseudodispozitiv?
   - ❑ `/dev/null`
   - ❑ `/dev/zero`
   - ❑ `/dev/hda3`
   - ❑ `/dev/urandom`
6. Care din următoarele dispozitive NU este asociat unui hard-disk?
   - ❑ `/dev/sda1`
   - ❑ `/dev/hda5`
   - ❑ `/dev/sda`
   - ❑ `/dev/ttyS0`
7. Care din următoarele se referă în mod direct la placa de bază?
   - ❑ southbridge
   - ❑ megabyte
   - ❑ GPU
   - ❑ cache
8. Care din următoarele acronime NU este corelat direct cu dispozitive fizice?
   - ❑ SCSI
   - ❑ USB
   - ❑ ACPI
   - ❑ TGZ
9. La care dispozitiv se referă acronimul AGP?
   - ❑ procesor
   - ❑ placa de rețea
   - ❑ placa grafică
   - ❑ hard-disk
10. La ce se referă în general noțiunea de "arhitectură pe N biți"?
    - ❑ dimensiunea registrelor
    - ❑ dimensiunea magistralei de date
    - ❑ dimensiunea memoriei cache L1
    - ❑ viteza maximă a southbridge-ului

Capitolul 8

Configurări de rețea

You know it's love when you memorize her IP address to skip DNS overhead.

Ce se învață din acest capitol?

- Noțiunea de rețea de calculatoare
- Topologia unei rețele de calculatoare
- Adresarea IP
- Parametrii unei rețele de calculatoare: adresă IP, mască de rețea, gateway
- Configurarea unei interfețe de rețea în Linux; adăugarea de rute în Linux
- Configurare temporară; configurare permanentă; configurare statică; configurare dinamică
- Configurare DNS
- Verificarea conectivității într-o rețea locală
- Configurarea rețelei în Windows
- Configurarea PPPoE în Linux

8.1 Concepte de rețea

Apariția și evoluția calculatoarelor au dus la dorința de a facilita comunicația între ele pentru a partaja date mai ușor sau pentru a putea îndeplini o sarcină comună. Deși dezvoltarea rețelelor de calculatoare a fost inițial lentă, implementarea lor a cunoscut o creștere mare după 1990, odată cu explozia numărului de calculatoare legate la Internet.

Dacă inițial rețelele de calculatoare au fost create pentru a facilita schimbul de informație în mediul academic, în momentul de față ele se adresează în mare măsură utilizatorilor privați și mediului comercial.

Pentru ca un calculator să fie conectat la o rețea trebuie realizate două conexiuni: o conexiune fizică și o conexiune logică.

Conexiunea fizică este utilizată pentru a codifica informația sub formă de semnale electrice (în cazul cel mai uzual întâlnit conexiunea se realizează prin cablu UTP), semnale optice (pentru legăturile prin fibra optică) sau semnale electromagnetice (pentru legăturile fără fir). În primele două cazuri ea se realizează printr-un cablu care leagă calculatorul de un echipament de rețea.

Conexiunea logică cuprinde un set de mecanisme de adresare complexe (ce asigură comunicația între calculatoarele conectate la rețea), cuprinde controlul fluxului de date și oferă garanția integrității datelor. Cel mai cunoscut mecanism de adresare prezent în cadrul conexiunii logice este IP (Internet Protocol). Acesta asigură identificarea unică și comunicarea calculatoarelor atât în rețelele locale, cât și în Internet.

8.1.1 Noțiuni de bază

Rețelele de calculatoare, din punctul de vedere al standardelor folosite, sunt împărțite în trei mari categorii: rețele locale de calculatoare (*Local Area Network* – LAN), rețele metropolitane de calculatoare (*Metropolitan Area Network* – MAN) și rețele de calculatoare pe arii extinse (*Wide Area Network* – WAN). În momentul de față această clasificare nu se mai păstrează 100%, rețelele MAN dispărând aproape complet și fiind înlocuite cu rețele LAN.

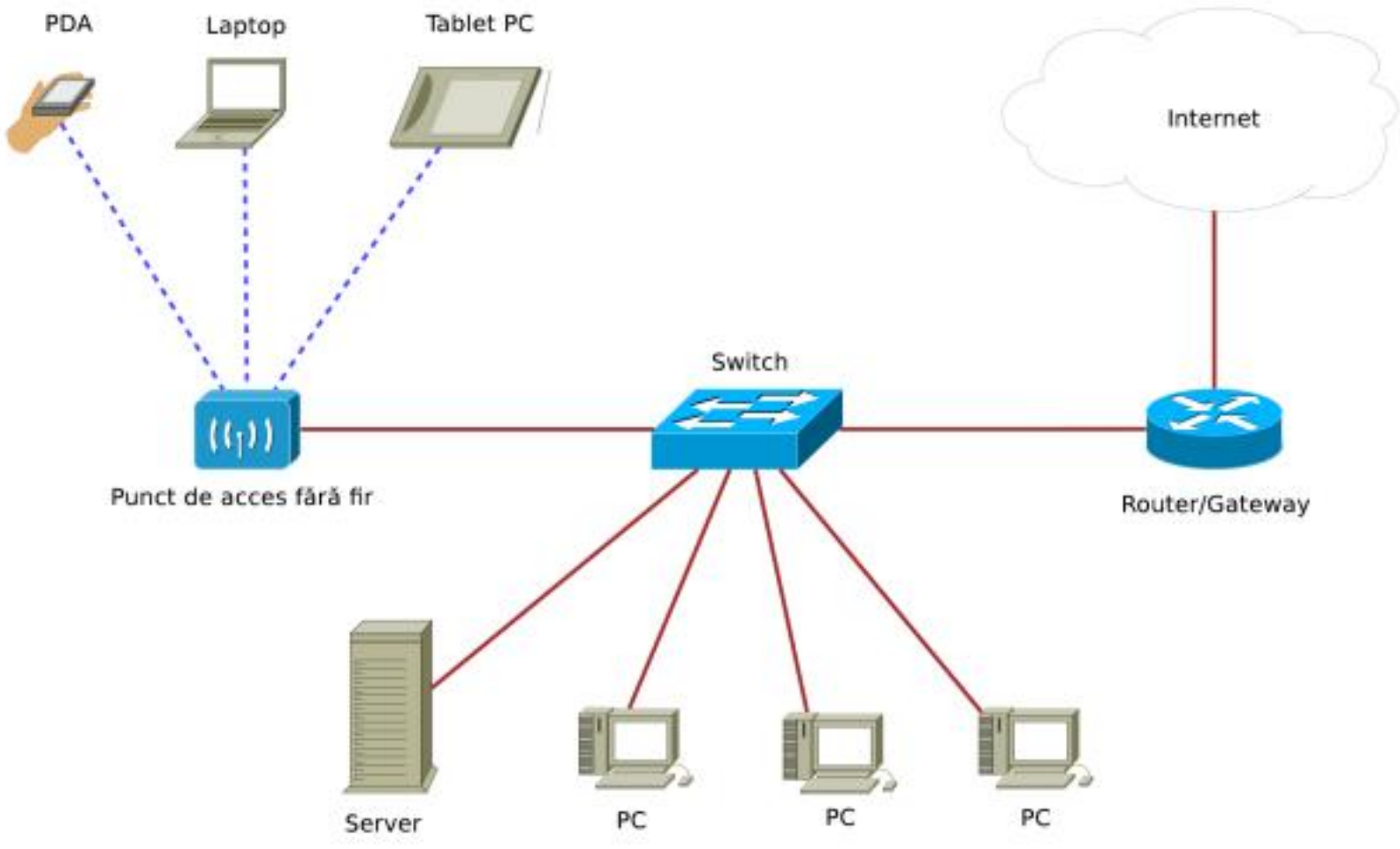


Figura 8.1: Schema unei rețele locale de calculatoare

O rețea locală de calculatoare este formată cel mai adesea din mai multe stații legate între ele. O stație poate fi un calculator, un laptop, un PDA, un *Tablet PC*, un server, un router etc.

Conectarea staților se realizează printr-un echipament de rețea numit switch. Acesta transferă informația între calculatoare bazându-se pe adresa MAC.

> **Adresa MAC** este un număr cu o lungime de 48 de biți care identifică în mod unic fiecare interfață de rețea Ethernet.

Prin urmare nu pot exista două interfețe de rețea Ethernet cu aceeași adresă MAC. Adresa MAC este înscrisă din fabrică într-o memorie ROM de pe placă și ea nu poate fi schimbată.

Datorită faptului că protocolul DHCP de configurare automată a adreselor IP utilizează frecvent adresa MAC pentru a aloca o adresă IP unui calculator, este uzuală practica de a modifica adresa MAC a unei plăci de rețea. Acest lucru nu înseamnă rescrierea memoriei ROM, ci specificarea unei alte adrese pe care sistemul de operare să o folosească în locul celei inițiale.

Switch-ul permite comunicația între calculatoare aflate în aceeași rețea locală. Pentru a putea accesa și alte calculatoare (cel mai adesea pentru a putea accesa Internetul) trebuie ca unul din echipamentele din rețea să aibă o a doua legătură la o altă rețea. Acest echipament poate să fie o stație sau un server cu mai multe interfețe de rețea sau un echipament de rețea dedicat numit **ruter**. Transferul informației din rețeaua locală mai departe înspre Internet se bazează pe adresa IP.

> **Adresa IP** este un număr cu o lungime de 32 de biți folosit pentru identificarea fiecărei interfațe de rețea în Internet.

O stație poate avea mai multe adrese IP, în mod uzual câte o adresă pentru fiecare interfață. Este posibil însă ca pe o interfață să fie configurate mai multe adrese IP.

8.1.2 IPv6

Până acum am folosit noțiunea IP pentru a ne referi la Internet Protocol version 4 (IPv4). Internet Protocol version 6 (IPv6) este următoarea generație de protocol de rețea, dezvoltat pentru a înlocui treptat IPv4.

> **Adresa IPv6** este un număr cu o lungime de 128 de biți folosit pentru identificarea fiecărei interfațe de rețea în Internet.

Principalul motiv pentru care a fost implementat un nou protocol de rețea pentru Internet este epuizarea adreselor IPv4. IPv6 are un spațiu de adrese mult mai mare decât IPv4 din cauza lungimii adreselor. Spațiul de adrese IPv6 oferă 2^128 adrese, față de IPv4 care are doar 2^32 adrese.

8.1.3 Moduri de adresare: adresare unicast, multicast, broadcast

Comunicațiile între echipamente pot fi împărțite în trei categorii, în funcție de numărul de destinatari:

- comunicații **unicast**: au o sursă și un destinatar;
- comunicații **multicast**: au o sursă și mai mulți destinatari ce fac parte dintr-un grup specific;
- comunicații **broadcast**: au o sursă și ca destinatar toate stațiile dintr-o rețea.

Atât mesajele de tip multicast cât și cele de tip broadcast sunt adresate mai multor destinatari simultan. Diferența dintre ele constă în faptul că mesajele de tip broadcast se adresează tuturor stațiilor dintr-o rețea, pe când cele multicast se adresează doar unora din aceste stații.

8.2 Parametri de rețea

8.2.1 Adresă IP și mască de rețea

Adresa IP pe 32 de biți a fost introdusă în perioada '80, când nu se preconiza creșterea spectaculoasă a Internetului. După 1990, datorită dezvoltării rețelelor, cerințele pentru adrese IP au crescut, astfel încât nu peste mult timp adresele au început să se epuizeze. Din acest motiv au apărut mai multe mecanisme menite să reducă numărul de adrese IP utilizate, printre care adresele IP private și translatarea de adrese IP (Network Address Translation – NAT). În paralel a fost dezvoltată și o soluție pe termen lung: protocolul IPv6, a cărui utilizare va crește în anii următori. În cadrul acestui protocol adresa IP are 128 de biți.

Pentru a ușura utilizarea adreselor IP, cei 32 de biți sunt separați în patru grupuri a câte 8 biți, fiecare grup fiind scris în formă zecimală. Astfel adresa IP utilizată în mod uzual este compusă din patru numere zecimale între 0 și 255 despărțite prin puncte:

```
01011001110101110001111010000111 - cei 32 de biti ai unei adrese IP
01011001.11010111.00011110.10000111 - patru grupuri a cate 8 biti
89.215.30.135 - fiecare grup este transformat in format zecimal
```

Atunci când o interfață de rețea are configurată o adresă IP, adresa are două componente: o parte din cei 32 de biți reprezintă adresa rețelei din care face parte stația și cealaltă parte a biților reprezintă adresa stației în cadrul rețelei. Sistemul este similar cu cel utilizat pentru codul poștal: în cadrul codului 014288, primele două cifre pot reprezenta județul, următoarele două pot reprezenta orașul, iar ultimele două pot reprezenta strada. Astfel destinația este localizată în arii din ce în ce mai restrânse.

Pentru a putea spune care dintre cei 32 de biți reprezintă adresa rețelei și care reprezintă adresa calculatorului în cadrul rețelei, este utilizată o mască de rețea. Aceasta este tot un număr de 32 de biți cu o caracteristică specială: este compusă, pornind de la stânga la dreapta, dintr-un bloc compact de 1 urmat de un bloc compact de 0. Pentru utilizarea mai ușoară a măștii de rețea este folosită aceeași notație ca și în cadrul adresei IP:

```
11111111000000000000000000000000 - cei 32 de biti ai unei masti
11111111.00000000.00000000.00000000 - patru grupuri a cate 8 biti
255.0.0.0 - fiecare grup este tranformat in format zecimal
```

Dacă în cadrul măștii de rețea un bit este egal cu 1, atunci bitul corespunzător din adresa IP face parte din *adresa rețelei*, iar dacă este egal cu 0, bitul corespunzător din adresa IP face parte din adresa calculatorului (stației) *în cadrul rețelei*:

```
01011001.11010111.00011110.10000111 - adresa IP
11111111.00000000.00000000.00000000 - masca de retea
<-A.R.-> <---------A.S.------------>
A.R.- Adresa retelei
A.S.- Adresa statiei
```

Pentru a putea afla din ce rețea face parte o adresă IP, se face ŞI logic pe biți între adresa IP și masca de rețea. După realizarea operației de ŞI logic se obține o adresa IP care are toți biții din partea de adresă a rețelei egali cu adresa rețelei din care face parte stația, și toți biții din partea de adresa a stației în cadrul rețelei egali cu 0:

```
01011001.11010111.00011110.10000111 = 89.215.30.135 - adresa IP
11111111.00000000.00000000.00000000 = 255.0.0.0 - masca de retea
----------------------------------- - SI logic pe biti
01011001.00000000.00000000.00000000 = 89.0.0.0 - adresa retelei
<-A.R.-> <-------A. C.------------>
01011001.11111111.11111111.11111111 = 89.255.255.255 - adr. broadcast
```

Adresa IP în care toți biții din partea de adresă a stației sunt egali cu 0 se numește **adresa rețelei** din care face parte stația. Toate stațiile dintr-o rețea locală au adrese IP care au aceeași adresă a rețelei. În felul acesta se realizează corespondența între conexiunea fizică la rețea și cea logică.

Adresa IP în care toți biții din partea de adresă a stației în cadrul rețelei sunt egali cu 1 se numește **adresa de broadcast** a rețelei. Dacă un pachet este trimis către această adresă, atunci el va fi procesat de toate calculatoarele din rețea.

Clase de adrese IP

Conceptul de clase de adrese IP este important pentru înțelegerea modului în care funcționează o rețea de calculatoare. Spațiul de adrese IP este împărțit în cinci categorii, denumite *clase de adrese*. În funcție de intervalul din care face parte primul octet al adresei IP (sau, echivalent, primul număr zecimal din notația cu punct), se stabilește clasa respectivei adrese IP:

- între 0 și 127, adresa IP face parte din clasa A,
- între 128 și 191, adresa IP face parte din clasa B,
- între 192 și 223, adresa IP face parte din clasa C,
- între 224 și 239, adresa IP face parte din clasa D,
- între 240 și 255, adresa IP face parte din clasa E.

Această împărțire a claselor de adrese a fost aleasă din considerente de performanță: pentru a determina clasa unei adrese IP oarecare, un dispozitiv hardware de rețea trebuie să inspecteze maximum primii patru biți ai adresei IP (valoarea biților notați cu - nu este importantă):

- 0 - - -. - - - -, adresa IP face parte din clasa A,
- 10 - -. - - - -, adresa IP face parte din clasa B,
- 110 -. - - - -, adresa IP face parte din clasa C,

- 1110. - - - -, adresa IP face parte din clasa D,
- 1111. - - - -, adresa IP face parte din clasa E.

Împărțirea pe clase determină două caracteristici importante ale unei adrese: tipul rutării (unicast/multicast/rezervat) și masca de rețea.

- Clasa A: 0.0.0.0 – 127.255.255.255 – masca 255.0.0.0 – unicast
- Clasa B: 128.0.0.0 – 191.255.255.255 – masca 255.255.0.0 – unicast
- Clasa C: 192.0.0.0 – 223.255.255.255 – masca 255.255.255.0 – unicast
- Clasa D: 224.0.0.0 – 239.255.255.255 – multicast
- Clasa E: 240.0.0.0 – 255.255.255.255 – rezervat

Modul acesta de clasificare a spațiului de adresă a fost utilizat într-o fază incipientă a Internetului. Din motive care țin de eficiența modului de utilizare a adreselor IP, în prezent, măștile de rețea nu au doar aceste trei lungimi fixe, ci pot avea lungime variabilă, iar împărțirea în cele cinci clase nu mai este întânită decât în cazuri rare.

Adrese IP publice și adrese IP private

În momentul în care numărul de adrese IP rămase libere a început să scadă simțitor, la începutul anilor '90, au apărut mecanisme menite să rezolve parțial această problema. Printre mecanismele apărute se numără și spațiul de adrese private.

Din fiecare clasă de adrese a fost rezervat un spațiu de adrese care nu pot fi accesate direct din afara rețelei locale. Practic stațiile care au configurate adrese IP private sunt invizibile din afara rețelei lor, și implicit din Internet. De aceea pot exista mai multe stații, făcând parte din rețele diferite, care au aceeași adresa IP privată. În felul acesta se realizează o economie mare de adrese IP.

Spațiile de adrese private sunt:

```
10.0.0.0 -- 10.255.255.255 din clasa A
170.16.0.0 -- 172.31.255.255 din clasa B
192.168.0.0 -- 192.168.255.255 din clasa C
```

Pentru ca o stație ce are configurată o adresă IP privată să poată accesa stații care nu se află în aceeași rețea cu ea este necesară utilizarea mecanismelor de NAT (*Network Address Translation*). De cele mai multe ori, prin NAT, toate stațiile dintr-o rețea cu adrese IP private vor fi recunoscute în afara rețelei ca având aceeași adresă IP publică.

8.2.2 Ruter implicit (default gateway)

Pentru a putea accesa calculatoare aflate în alte rețele, este necesar ca unul dintre echipamentele conectate în rețea să aibă o a doua interfață conectată la o altă rețea. Din cea de-a doua rețea se poate realiza mai departe accesul către Internet. Acest echipament se numește *gateway*.

Atunci când o stație A dorește să comunice cu o stație B, primul pas pe care îl face este să verifice dacă adresa IP a lui B este în aceeași rețea cu adresa lui A. În cazul în care

cele două stații sunt în aceeași rețea, informațiile sunt trimise direct către B. Dacă B nu se află în aceeași rețea cu A, atunci informațiile sunt trimise către gateway, urmând ca acesta să găsească o rută către B.

De exemplu când A1 vrea să comunice cu A3, fiind amandouă în aceeași rețea locală, îi va trimite informațiile direct. Calea pachetelor este reprezentată în figura 8.2.

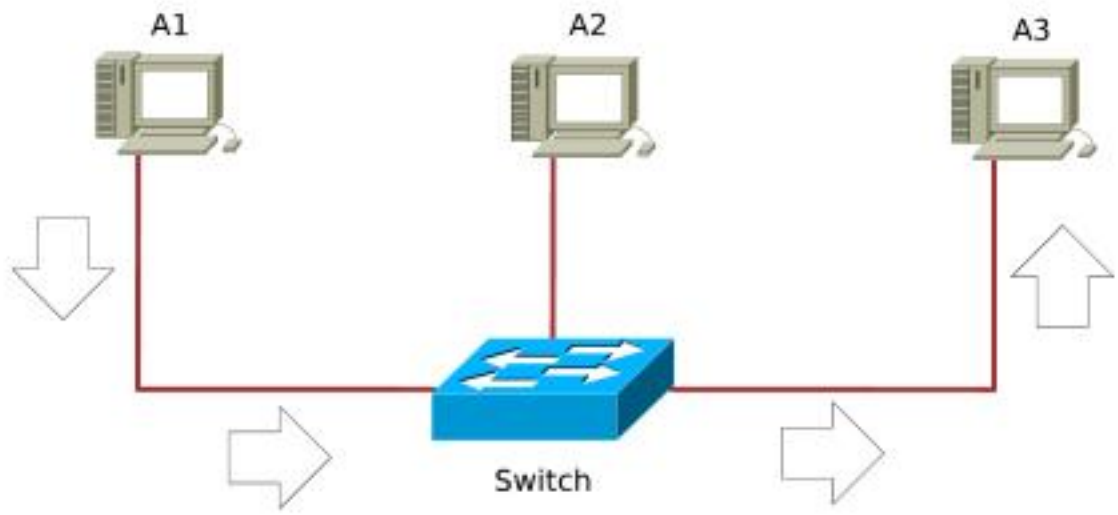


Figura 8.2: Comunicarea în rețeaua locală

În cazul în care A1 vrea să comunice cu B2, va trimite informațiile către R1 care este gateway-ul. R1 va verifica dacă rețeaua destinație este direct conectată sau dacă cunoaște calea către ea. Schema din figura 8.3 reprezintă calea pachetelor de la A1 la B2:

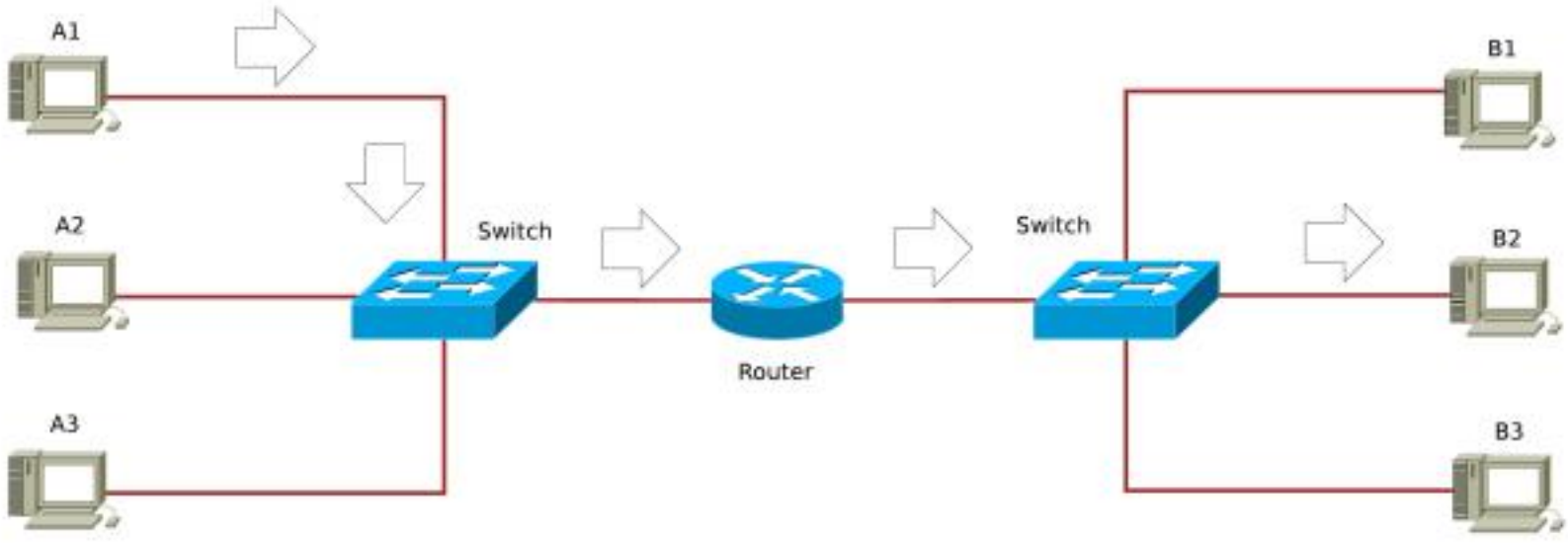


Figura 8.3: Comunicarea în afara rețelei locale

Prin urmare este necesar ca fiecare stație din rețea să cunoască adresa gateway-ului pentru a putea avea acces în afara rețelei și masca de rețea pentru a vedea dacă este cazul să folosească gateway-ul sau nu.

8.2.3 DNS

Un utilizator accesează în mod normal un număr mare de servicii de rețea, cele mai multe fiind oferite de diverse servere. Accesarea fiecărui serviciu este condiționată de cunoașterea adresei IP a serverului care oferă serviciul respectiv. Spre exemplu, pentru a accesa o pagină web este necesară cunoașterea adresei IP a serverului pe care este găzduită acea pagină.

Pentru că memorarea unui număr mare de adrese IP nu este un lucru comod, a apărut serviciul numit Sistemul numelor de domenii (Domain Name System – DNS). În cadrul acestui serviciu se realizează corespondența între un șir de caractere și o adresă IP. De aceea DNS-ul poate fi privit ca o carte de telefon de unde, dacă se cunoaște numele unui server, i se poate afla adresa IP.

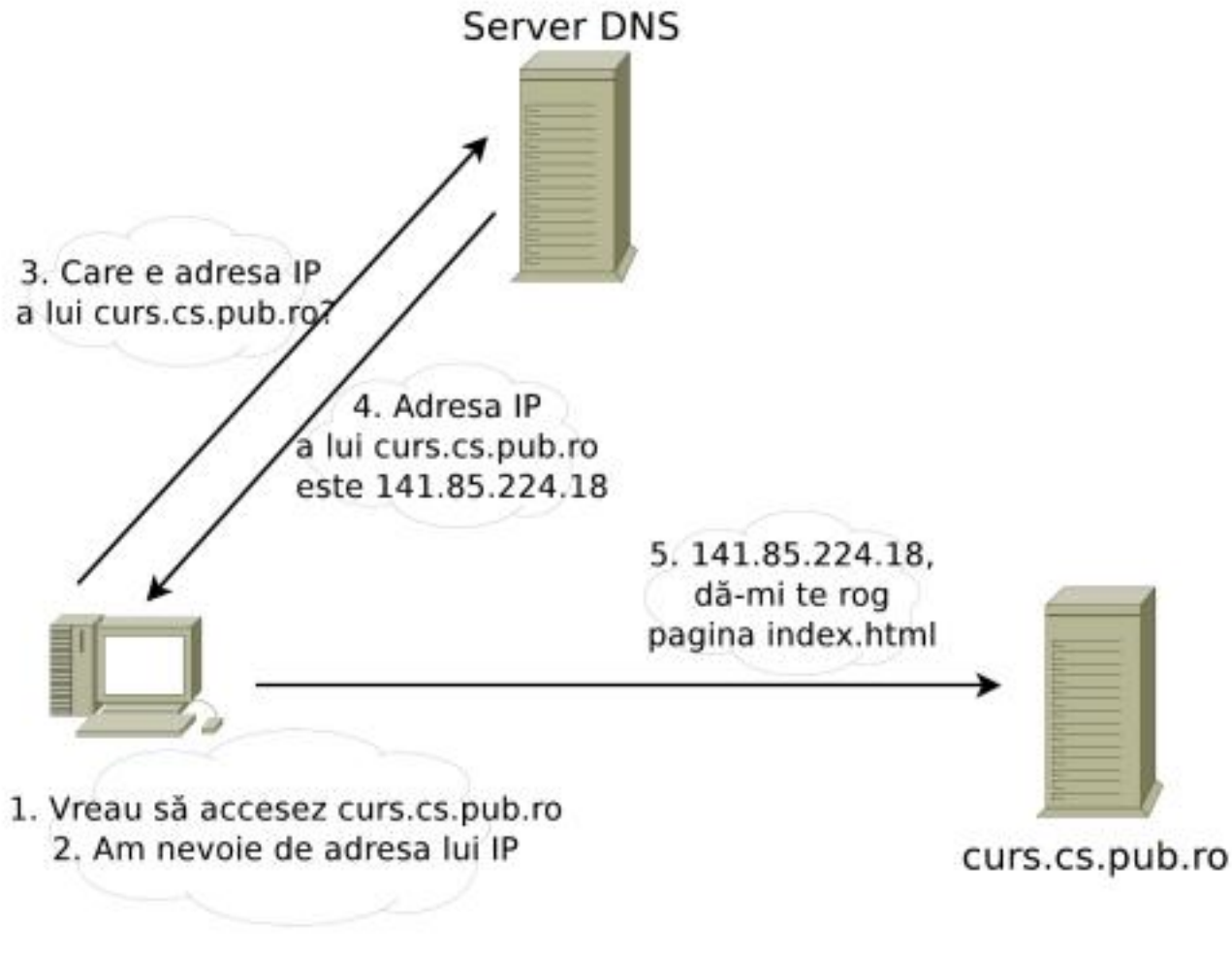


Figura 8.4: Rolul unui server DNS

Un nume de domeniu (asociat unui calculator) este compus din mai multe părți despărțite prin puncte: de exemplu curs.cs.pub.ro. Cea din extremitatea dreaptă se numește *Top Level Domain – TLD* și poate conține indicativul unei țări (.ro pentru România, .us pentru SUA, .jp pentru Japonia etc.) sau poate reprezenta un anumit tip de organizație (de exemplu .com pentru organizațiile comerciale, .org pentru organizațiile non-profit, .mil pentru armata SUA etc).

În stânga Top Level Domain se găsesc subdomenii. În exemplul de mai sus subdomeniul pub este asociat Universității Politehnica București, cs este asociat Facultății de Automatică și Calculatoare, Catedra de Calculatoare.

Serviciile de DNS sunt oferite de servere specializate. Pentru a putea accesa un astfel de server, este necesară cunoașterea adresei lui IP. Exista servere publice care oferă servicii de DNS, însă de cele mai multe ori este de preferat ca în cadrul rețelei locale să existe un server de DNS din motive ce țin de timpul de răspuns.

Chiar dacă serverul de DNS este plasat în rețeaua locală, DNS-ul rămâne cel mai lent serviciu de rețea și componenta cu cel mai mare timp de răspuns în mecanismul de comunicație în rețea. De aceea se spune că "*You know it's love when you memorize her IP address to skip DNS overhead*".

Este posibil uneori ca adresa IP asociată unei stații să fie dinamică. Aceasta înseamnă că din timp în timp stația respectivă va primi o altă adresă IP. Pentru a putea accesa stația fără să i se cunoască adresa IP existentă la un moment dat se folosește sistemul

Tabelul 8.1: Configurarea rețelei

	Configurare statică	**Configurare dinamică**
Configurare temporară	**`ifconfig`**	**`dhclient`, `dhcpcd`**
Configurare permanentă	`/etc/network/interfaces`	`/etc/network/interfaces`

numit DynDNS (*Dynamic DNS*). În cadrul acestui sistem stația va trimite către un server DNS specializat informații actualizate legate de adresa ei IP.

8.3 Configurări temporare

8.3.1 Interfețe de rețea. Configurări permanente și configurări temporare

Din punct de vedere logic, fiecărei plăci de rețea îi corespunde în cadrul sistemului de operare o interfață. Pe această interfață se configurează o adresă IP și o mască de rețea.

În Linux interfețele de rețea conțin în denumirea lor două componente. Prima parte reprezintă tipul interfeței (de exemplu `eth` pentru Ethernet, `wlan` pentru placile de rețea fără fir etc) și numărul ei (`eth0` reprezintă prima interfață de Ethernet, `wlan1` reprezintă a două interfață fără fir etc).

Configurarea unei adrese IP se poate realiza în două moduri: temporară sau permanentă. Fiecare din cele două se poate realiza la rândul ei static (parametrii sunt introduși manual de administrator) sau dinamic (parametrii se configurează în mod automat).

Configurarea temporară statică se realizează cu ajutorul comenzii **`ifconfig`** (a cărei denumire vine de la *interface configuration*). Efectul acestei configurări este imediat (parametrii se aplica imediat după ce comanda este dată) însă odată cu repornirea sistemului de operare, configurările se pierd.

Configurarea temporară dinamică se realizează cu ajutorul unui client DHCP, de exemplu **`dhclient`** sau **`dhcpcd`**.

Configurarea permanentă statică sau dinamică se realizează cu ajutorul fișierului `/etc/network/interfaces`. Efectul acestei comenzi nu este imediat (configurările nu se aplică odată cu salvarea fișierului), însă configurările se păstrează și după repornirea sistemului de operare. Acest lucru se petrece deoarece fișierul în care sunt trecute configurările este citit și interpretat de sistemul de operare la inițializare, configurările din fișier fiind din nou aplicate.

Şi configurarea unui gateway se poate realiza în mod temporar cu ajutorul comenzii route sau în mod permanent prin fișierul de configurare `/etc/network/interfaces`.

Fiecare din cele două metode (permanent vs. temporar) are avantajele sale. Configurările temporare au efect imediat și se aplică ușor. Cele permanente se pot

aplica automat la fiecare inițializare a sistemului de operare.

8.3.2 Configurarea temporară statică a unei adrese IP pe o interfață

Pentru a configura o interfață de rețea în mod temporar se folosește comanda **`ifconfig`**. Modul de utilizare al comenzii este următorul:

```
root@ubuntu:~# ifconfig [-v] [-a] [-s] [interface]
root@ubuntu:~# ifconfig [-v] interface [aftype] options | address ...
```

Prin utilizarea comenzii fără niciun parametru se pot afla informații despre interfețele de rețea active:

```
root@asgard:/home/george# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          inet6 addr: fe80::214:d1ff:fe38:73a6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500 Metric:1
          RX packets:18982017 errors:0 dropped:0 overruns:0 frame:0
          TX packets:27864548 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:144959484 (130 MiB) TX bytes:21665362 (20.6 MiB)
          Interrupt:18 Base address:0x4400

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:16436 Metric:1
          RX packets:5777 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5777 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:360919 (352.4 KiB) TX bytes:360919 (352.4 KiB)
```

Pe lângă interfețele corespunzătoare plăcilor de rețea există și interfața `lo`. Numele acestei interfețe vine de la *loopback*. Orice informație transmisă pe loopback se va întoarce înapoi tot pe loopback.

Interfața de loopback este o interfață virtuală (fără un corespondent fizic) care este folosită în scopuri de testare. Ea are, deobicei, asociată o adresă IP standard: `127.0.0.1`. Dacă un calculator nu are nicio placă de rețea, interfața de loopback va fi singura existentă.

Parametrii uzuali ai comenzii **`ifconfig`** sunt următorii:

- `-v` afișează informații detaliate în cazul erorilor;
- `-a` afișează informații despre toate interfețele existente, fie că sunt active sau inactive.

```
root@asgard:/home/george# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]

```

```
root@asgard:/home/george# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
eth1      Link encap:Ethernet  HWaddr 00:18:F3:AB:45:62
          BROADCAST MULTICAST  MTU:1500  Metric:1
          [...]
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]
```

- `-s` afișează statistici legate de interfețele de rețea în format compact, sub forma unui tabel:

```
root@asgard:/home/george# ifconfig -s
iface   MTU Met  RX-OK RX-ERR  [...]  TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0   1500 0   193488      0  [...] 282347      0      0      0 BMRU
lo    16436 0     5849      0  [...]    584      0      0      0 LRU
```

- `interface` reprezintă numele interfeței despre care se doresc informații sau numele interfeței care va fi configurată. De exemplu, pentru a afișa informații despre interfața de rețea eth0 se folosește comanda

```
root@asgard:/home/george# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
```

- `address` reprezintă adresa IP care va fi configurată pe interfață. De exemplu, pentru a configura adresa `IP 10.1.1.3` pe interfața `eth0`:

```
root@asgard:/home/george# ifconfig eth0 10.1.1.3

root@asgard:/home/george# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.3  Bcast:10.255.255.255  Mask:255.0.0.0
          [...]
```

- `options` reprezintă opțiuni avansate ce pot fi configurate

Opțiunile cele mai des folosite sunt următoarele:

- `down` dezactivează o interfață. Odată dezactivată, interfața nu va mai trimite și nu va mai primi niciun pachet

```
root@asgard:/home/george# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]

root@asgard:/home/george# ifconfig eth0 down

root@asgard:/home/george# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]
```

Se poate observa faptul că, după ce interfața a fost dezactivată, ea nu mai apare în lista de interfețe active. Pentru a afișa toate interfețele existente se folosește parametrul `-a`.

- `up` este utilizată pentru a activa o interfață. În cazul în care este specificată o adresă IP pentru a fi configurată pe interfață, opțiunea `up` este considerată implicit.

```
root@asgard:/home/george# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]

root@asgard:/home/george# ifconfig eth0 up

root@asgard:/home/george# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.2  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]
```

- `netmask addr` specifică masca de rețea asociata interfeței. În cazul în care această opțiune nu este prezentă se va considera masca implicită clasei IP-uri din care face parte adresa IP a interfeței Se poate observa din exemplele anterioare faptul că, nespecificând masca de rețea, aceasta a fost considerată implicit `255.0.0.0`, care este valoarea pentru clasa A de adrese IP din care face parte `10.1.1.3`. În cazul în care se specifică și valoarea măștii ca opțiune, rezultatul va fi:

```
root@asgard:/home/george# ifconfig eth0 10.1.1.3 netmask
255.255.255.0

root@asgard:/home/george# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.3  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
```

- `hw class addr` specifică utilizarea unei alte adrese MAC și nu cea din memoria ROM a plăcii de rețea, în cazul în care driverul plăcii de rețea oferă suport pentru acest lucru. Este foarte important ca interfața de rețea să fie dezactivată înaintea configurării unei noi adrese MAC:

```
root@asgard:/home/george# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet addr:10.1.1.3  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]

root@asgard:/home/george# ifconfig eth0 down

root@asgard:/home/george# ifconfig eth0 hw ether 01:02:03:04:05:06
up

root@asgard:/home/george# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 01:02:03:04:05:06
          inet addr:10.1.1.3  Bcast:10.1.1.255  Mask:255.255.255.0
          [...]
```

Utilitatea folosirii unei adrese MAC diferită de cea a plăcii de rețea apare datorită faptului că protocolul DHCP de configurare automată a adreselor IP utilizează frecvent adresa MAC pentru a aloca o adresă IP unui calculator. Pentru a obține o anumită adresă IP, trebuie folosită adresa MAC asociată în cadrul serverului DHCP cu adresa IP dorită.

8.3.3 Asigurarea conectivității la Internet. Configurarea temporară statică a unei rute implicite

Comanda **`ifconfig`** permite doar configurarea unei adrese IP pe o interfață. Pentru a putea accesa Internetul este nevoie să fie specificată adresa unui gateway.

Specificarea temporară a unui gateway se realizează cu ajutorul unei rute. O **rută** reprezintă calea pe care o urmează pachetele în drumul lor către destinație. Rutele sunt reținute în cadrul unui structuri numite **tabelă de rutare**. Aceasta conține practic informații despre toate destinațiile accesibile la un moment dat.

Printre informațiile reținute pentru fiecare rută se numără: adresa destinației (poate fi adresa unei stații sau adresa unei întregi rețele), masca de rețea asociată și adresa echipamentului de rețea către care se trimit informațiile pentru a ajunge la destinație.

Configurarea tabelei de rutare se realizează cu ajutorul comenzii route. Aceasta permite adăugarea, modificarea și ștergerea de rute. Pentru a vedea care sunt rutele existente la un moment dat (ca atare pentru a vedea conținutul tabelei de rutare) se utilizează comanda **`route`** fără parametri:

```
root@asgard:/home/george# route
Kernel IP routing table
Destination   Gateway      Genmask         Flags Metric Ref  Use Iface
localnet      *            255.255.255.0   U     0      0      0 eth0
```

În rezultatul de mai sus se vede faptul că singura rută existentă este cea asociată rețelei locale (`localnet`) .Această rută (care este practic asociată cu rețeaua locală direct conectată) este introdusă în tabela de rutare după ce interfața de rețea asociată rețelei este configurată.

Formatul comenzii utilizat pentru adăugarea unei rute implicite este următorul:

```
root@@ubuntu:~# route add default gateway addr
```

Configurarea unei rute implicite se realizează astfel:

```
root@asgard:/home/george# route add default gateway 10.1.1.1

root@asgard:/home/george# route
Kernel IP routing table
Destination   Gateway      Genmask         Flags Metric Ref  Use Iface
localnet      *            255.255.255.0   U     0      0      0 eth0
default       10.1.1.1     0.0.0.0         UG    0      0      0 eth0
```

Unul dintre parametrii utili ai comenzii route este `-n` (*numeric*) Folosind acest parametru adresele din tabela de rutare vor fi afișat în format numeric în loc să se afișeze numele serverelor sau stațiilor. Este folosit atunci când se dorește interogarea unui server DNS (de exemplu atunci când nu este configurat un astfel de server).

```
root@asgard:/home/george# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.1.0        *               255.255.255.0   U     0      0        0 eth0
0.0.0.0         10.1.1.1        0.0.0.0         UG    0      0        0 eth0
```

8.4 Configurări permanente

8.4.1 Configurarea permanentă a unei interfețe de rețea

Configurările de rețea permanente se realizează în cadrul fișierului `/etc/network/interfaces`. Manualul de utilizare al acestui fișier se poate accesa cu comanda:

```
man interfaces
```

Un exemplu de fișier de configurare este următorul:

```
root@asgard:/home/george# cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo eth0
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
iface eth0 inet static
        address 10.1.1.2
        netmask 255.255.255.0
        network 10.1.1.0
        broadcast 10.1.1.255
        gateway 10.1.1.1
        dns-nameservers 10.1.1.1
```

Fișierul `/etc/network/interfaces` conține diverse configurări ale interfețelor de rețea, configurări care sunt interpretate de utilitarele **`ifup`** și **`ifdown`**. **`ifup`** este un utilitar ce permite activarea unei interfețe de rețea. El este similar opțiunii up a comenzii **`ifconfig`**. **`ifdown`** dezactivează o interfață de rețea, fiind similar cu opțiunea `down` a comenzii **`ifconfig`**.

Pentru a activa toate interfețele definite cu `auto` în `/etc/network/interfaces` se folosește următoarea comandă:

```
root@ubuntu:~# ifup -a
```

Pentru a activa o anumită interfață se precizează numele acesteia, de exemplu:

```
root@ubuntu:~# ifup eth0
```

Pentru a dezactiva toate interfețele active se folosește următoarea comandă:

```
root@ubuntu:~# ifdown -a
```

În cadrul fișierului `interfaces` se găsesc mai multe declarații de tipul `iface`, `auto` sau `allow`.

Declarațiile `auto` specifică ce interfețe sunt activate atunci când **`ifup`** este folosit împreună cu parametrul `-a`. Un astfel de apel (**`ifup -a`**) este folosit la pornirea sistemului de operare. Pornirea unei interfețe marcate ca `auto` este realizată în momentul în care `udev` (vezi secțiunea 7.5) a creat mecanismele necesare funcționării interfeței.

Declarațiile `allow` specifică interfețele care sunt activate automat de către diverse subsisteme. Astfel activarea nu mai este realizată după ce `udev` a detectat placa de rețea, ci în momentul în care aceasta devine disponibilă.

Pentru specificarea unei interfețe de loopback se folosește următoarea linie:

```
1 iface lo inet loopback
```

După declarația `iface` urmează numele interfeței, apoi tipul de adrese pe care interfața îl folosește (`inet` reprezintă adresare IP, `inet6` reprezintă adresare IPv6). Pentru a marca o interfață ca loopback, se folosește parametrul `loopback`.

O linie asemănătoare este folosită și pentru a specifică orice alta interfață existentă. Spre exemplu, pentru configurarea interfeței `eth0`:

```
1 iface eth0 inet static
```

După declarația `iface` urmează numele interfeței apoi tipul de adrese pe care îl folosește interfața, iar la final, prin includerea parametrului `static` se specifică faptul că interfața este configurată manual de administrator, rămânând fixă până la următoarea schimbare manuală. O altă metodă de configurare este cea în care se folosește DHCP, prin care stația cere configurările interfeței de la un server DHCP.

Dacă parametrul `static` a fost precizat în declarația `iface`, atunci după această declarație trebuie să urmeze parametrii cu care va fi configurată interfața de rețea:

```
1 iface eth0 inet static
2         address 10.1.1.2
3         netmask 255.255.255.0
4         network 10.1.1.0
5         broadcast 10.1.1.255
6         gateway 10.1.1.1
7         dns-nameservers 10.1.1.1
```

`address` specifică adresa IP a interfeței (parametru obligatoriu), `netmask` specifică masca de rețea care va fi folosită (parametru obligatoriu), `network` specifica adresa rețelei din care face parte interfața (parametru obligatoriu doar pentru kernelul 2.0.x, adresă ce se poate obține făcând ŞI logic pe biți între adresa IP și masca de rețea, `broadcast` specifică adresa de broadcast a rețelei din care face parte interfața (parametru opțional).

După modificarea fișierului `/etc/network/interfaces` este necesară restartarea serviciului de rețea pentru a citi noua configurație:

```
1 root@ubuntu:~# /etc/init.d/networking restart
```

Serviciul de rețea poate fi oprit folosind argumentul `stop` și pornit folosind argumentul `start`:

```
root@ubuntu:~# /etc/init.d/networking stop

root@ubuntu:~# /etc/init.d/networking start
```

8.4.2 Configurarea permanentă statică a unei rute implicite

O a doua modalitate de specificare a rutei implicite este folosirea fișierului `/etc/network/ interfaces`. Structura acestuia a fost descrisă în paragraful anterior.

Printre parametrii care se pot configura după declarația `iface` se află și gateway-ul implicit. Specificarea acestuia se realizează astfel:

```
iface eth0 inet static
        [...]
        gateway 10.1.1.1
```

unde `10.1.1.1` este adresa IP a gateway-ului.

8.4.3 DHCP. Configurarea unei interfețe în mod automat

Configurarea parametrilor unei interfețe de rețea se poate realiza static sau dinamic.

În cazul configurărilor statice toți parametrii trebuie introduși manual pe fiecare stație din rețea. În cazul în care se schimbă gateway-ul, de exemplu, pe fiecare stație din rețea trebuie realizată această schimbare.

Configurările dinamice permit atribuirea automată a parametrilor de rețea, printre care adresa IP, masca de rețea, gateway-ul, serverul de DNS.

Cel mai utilizat protocol de configurare dinamică (automată) este **DHCP** (*Dynamic Host Configuration Protocol*). Pentru ca acest protocol să funcționeze este necesar ca în rețeaua locală să se găsească un server de DHCP. Serverului îi sunt specificate plaja de adrese IP pe care le poate atribui calculatoarelor și restul configurărilor de rețea.

Atunci când o stație dorește să obțină o adresă IP, va trimite un mesaj către toată rețeaua (broadcast) prin care încearcă să determine dacă există un server de DHCP. Dacă acest server există, atunci el va trimite configurațiile către stație.

Configurarea parametrilor interfeței de rețea folosind DHCP se poate realiza în mod temporar sau permanent.

Pentru a realiza o configurare temporară se pornește clientul de DHCP, de exemplu cu ajutorul comenzii **`dhclient`**:

```
root@asgard:/home/george# dhclient
Internet Systems Consortium DHCP Client V3.0.6
Copyright 2004-2007 Internet Systems Consortium.
All rights reserved.
For info, please visit http://www.isc.org/sw/dhcp/

Listening on LPF/eth0/00:40:f4:cc:c5:41
Sending on   LPF/eth0/00:40:f4:cc:c5:41
```

```
Sending on   Socket/fallback
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 6
DHCPOFFER from 10.1.1.1
DHCPREQUEST on eth0 to 255.255.255.255 port 67
DHCPACK from 10.1.1.1
bound to 10.1.1.2 -- renewal in 815 seconds.
```

Interfața va rămâne astfel configurată până la următoarea repornire a stației sau până când configurările vor fi schimbate.

Pentru a realiza o configurare permanentă prin DHCP se folosește de asemenea fișierul `/etc/network/interfaces`. Linia care configurează interfața eth0 arată astfel:

```
iface eth0 inet dhcp
```

8.4.4 DNS. Configurarea serverelor de DNS

Fișiere de configurare

Pentru a configura adresele serverelor de DNS se folosește fișierul `/etc/resolv.conf`. În cadrul acestui fișier se pot specifica adresele mai multor servere de DNS. În cazul în care unul din ele nu conține informații despre domeniul căutat, se va încerca interogarea celorlalte.

Manualul de utilizare al fișierului `/etc/resolv.conf` se poate accesa folosind comanda:

```
man resolv.conf
```

Un exemplu de fișier `/etc/resolv.conf` este următorul:

```
search localdomain
nameserver 10.1.1.1
nameserver 217.115.138.24
nameserver 128.107.241.185
```

În cadrul fișierului cea mai importantă declarație este declarația nameserver. Ea definește adresa unui server de DNS. Fișierul poate conține mai multe declarații de acest tip.

Aplicarea configurărilor se realizează imediat deoarece fișierul `resolv.conf` este interogat de fiecare proces atunci când dorește rezolvarea unui nume de domeniu.

Deși nu este uzual, serverele de DNS se pot configura și din cadrul fișierului `/etc/network/interfaces`.

De cele mai multe ori, atunci când o interfață este configurată automat prin DHCP, serverul de DHCP oferă pe lângă adresa IP și adresa unui server DNS, care va fi salvată automat în `/etc/resolv.conf`.

Utilitarul `resolvconf`

Folosind utilitarul **`resolvconf`** se pot afișa informații despre serverele de nume configurate în sistem și se poate modifica configurația din fișierul

/etc/resolv.conf. Folosind opțiunea -a se poate modifica fișierul /etc/resolv.conf:

```
ubuntu@ubuntu:~$ cat /etc/resolv.conf
nameserver 192.168.2.1

ubuntu@ubuntu:~$ echo "nameserver 192.168.2.100" | resolvconf -a eth0

ubuntu@ubuntu:~$ echo "nameserver 192.168.2.101" | resolvconf -a eth1

ubuntu@ubuntu:~$ cat /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by
resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 192.168.2.100
nameserver 192.168.2.101
```

O diferență între configurarea cu **resolvconf** și modificarea directă a fișierului /etc/ resolv.conf este specificarea interfeței pentru care se face această configurație. Pentru a șterge configurația asociată cu o anumită interfață se folosește următoarea comandă:

```
ubuntu@ubuntu:~$ resolvconf -d eth0

ubuntu@ubuntu:~$ cat /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by
resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 192.168.2.101
```

Interogarea serverelor de nume

Folosind comanda **host** se pot obține informații despre un anumit domeniu, cum ar fi adresa IP asociată, serverul de nume al domeniului, serverul de mail, descrierea, alias-ul etc.

De exemplu, adresa IP asociată se poate afla folosind următoarea comandă:

```
ubuntu@ubuntu:~$ host -t A cs.pub.ro
cs.pub.ro has address 141.85.37.5
```

Pentru a obține serverul de nume asociat cu acest domeniu se va folosi comanda:

```
ubuntu@ubuntu:~$ host -t NS cs.pub.ro
cs.pub.ro name server ns.cs.pub.ro.
cs.pub.ro name server pub.pub.ro.
```

Serverul de mail se poate obține folosind comanda următoare:

```
ubuntu@ubuntu:~$ host -t MX cs.pub.ro
cs.pub.ro mail is handled by 5 mail.cs.pub.ro.
```

Comanda următoare va afișa descrierea domeniului:

```
ubuntu@ubuntu:~$ host -t TXT cs.pub.ro
cs.pub.ro descriptive text "Politehnica University of Bucharest,
Computer Science Departament"
```

8.4.5 Aliasuri. /etc/hosts

Aliasurile reprezintă etichete care sunt asociate unor adrese de IP. Sistemul de aliasuri funcționează asemănător cu cel de DNS: de exemplu, atunci când o stație dorește să acceseze serverul numit `my-print-server`, ea va încerca să găsească adresa IP a serverului respectiv.

Există însă două deosebiri majore față de DNS. Prima constă în faptul ca etichetele asociate unor adrese IP nu trebuie să respecte convențiile numelor de domenii DNS (de exemplu nu trebuie să se termine cu un Top Level Domain). A doua deosebire constă în felul în care este găsită adresa IP asociată unei etichete. Dacă în cazul DNS se folosesc servere dedicate, în cazul aliasurilor este interogat fișierul `/etc/hosts`. În acest fișier sunt trecute corespondențele între etichete și adrese IP.

Un exemplu de fișier hosts este următorul:

```
root@asgard:/home/george# cat /etc/hosts
127.0.0.1 localhost asgard
127.0.1.1 asgard

#Servers
10.1.1.100 print.mydomain.ro my-print-server
10.1.1.101 ftp.mydomainro my-ftp-server
192.168.1.150 mail-server

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

Fișierul hosts conține mai multe linii de forma:

```
adresa_IP numele_statiei [eticheta] [eticheta] ...
```

Câmpul cel mai din stânga este adresa de IP. Urmează apoi numele principal al stației, și etichetele care se asociază cu această adresă, despărțite prin spațiu.

De obicei, atunci când sunt configurate și servere de DNS și aliasuri, aliasurile au prioritate și vor fi verificate primele.

8.4.6 Configurarea numelui stației curente

Pentru stația curentă se poate configura un nume local. Folosind oricare dintre următoarele comenzi se poate afla numele stației:

```
ubuntu@ubuntu:~$ hostname

ubuntu@ubuntu:~$ uname -n

ubuntu@ubuntu:~$ cat /proc/sys/kernel/hostname

ubuntu@ubuntu:~$ sysctl kernel.hostname
```

Numele stației se poate modifica temporar folosind comanda **hostname**, ca în exemplul următor:

```
ubuntu@ubuntu:~$ hostname
ubuntu

ubuntu@ubuntu:~$ hostname uso

ubuntu@ubuntu:~$ hostname
uso
```

Pentru a realiza o configurare permanentă se va modifica fișierul `/etc/hostname` care conține numele stației. Pentru a determina recitirea configurației din acest fișier se va rula următorul script:

```
ubuntu@ubuntu:~$ /etc/init.d/hostname.sh
```

8.5 Testarea configurațiilor de rețea

Odată ce s-au realizat configurațiile de rețea, acestea trebuie testate. Pentru testarea lor sunt folosite în mod uzual două utilitare: **ping** și **traceroute**. Aceste utilitare sunt folosite și pentru a depista problemele uzuale care pot apărea în funcționarea unei rețele.

8.5.1 Ping

Utilitarul **ping** este folosit pentru a verifica conectivitatea între două stații. El testează conexiunea fizică și parte din cea logică, folosindu-se de protocoalele IP și ICMP (*Internet Control Message Protocol*) pentru a trimite și pentru a primi mesaje.

Ping primește adresa IP a stației destinație și va trimite către aceasta un mesaj `ECHO REQUEST`. Atunci când acest mesaj este recepționat, stația destinație va răspunde cu un mesaj similar, `ECHO REPLY`. Dacă răspunsul destinației ajunge înapoi la sursă, conexiunea dintre ele funcționează. În caz contrar există probleme care împiedică cele două stații că comunice.

Principalii parametri ai comenzii ping sunt următorii:

```
ping [-f] [-c count] [-i interval] [-I interface] destination
```

Fără nicio opțiune, **ping** va trimite un număr nelimitat de mesaje către destinație, până când va fi întrerupt:

```
root@asgard:/home/george# ping 10.1.1.1
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data.
64 bytes from 10.1.1.1: icmp_seq=1 ttl=255 time=0.395 ms
64 bytes from 10.1.1.1: icmp_seq=2 ttl=255 time=0.359 ms
[CTRL+C]
--- 10.1.1.1 ping statistics ---
2 packets transmitted, 2 received, 0\% packet loss, time 8998ms
rtt min/avg/max/mdev = 0.341/0.400/0.707/0.106 ms
```

Opțiunea `-c` specifică numărul de pachete trimise, iar opțiunea `-i` specifică intervalul în secunde între două pachete trimise:

```
root@asgard:/home/george# ping 10.1.1.1 -c 3 -i 1
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data.
64 bytes from 10.1.1.1: icmp_seq=1 ttl=255 time=0.326 ms
64 bytes from 10.1.1.1: icmp_seq=2 ttl=255 time=0.459 ms
64 bytes from 10.1.1.1: icmp_seq=3 ttl=255 time=0.361 ms
--- 10.1.1.1 ping statistics ---
3 packets transmitted, 3 received, 0\% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.326/0.382/0.459/0.056 ms
```

Opțiunea `-I` specifică pe ce interfață se vor trimite pachetele:

```
root@asgard:/home/george# ping 10.1.1.1 -I eth0 -c 3
PING 10.1.1.1 (10.1.1.1) from 10.1.1.2 eth0: 56(84) bytes of data.
64 bytes from 10.1.1.1: icmp_seq=1 ttl=255 time=0.331 ms
64 bytes from 10.1.1.1: icmp_seq=2 ttl=255 time=0.326 ms
64 bytes from 10.1.1.1: icmp_seq=3 ttl=255 time=0.367 ms

--- 10.1.1.1 ping statistics ---
3 packets transmitted, 3 received, 0\% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.326/0.341/0.367/0.023 ms
```

Opțiunea `-f` permite transmiterea unui număr nelimitat de pachete `ECHO REQUEST`, fără să se aștepte răspunsul `ECHO REPLY`. În acest fel destinația este inundată (flood) de mesaje. Datorită potențialului pe care îl are această opțiune de a fi folosită în atacuri DoS (vezi secțiunea 10.4.1), parametrul `-f` poate fi folosit doar de către `root`:

```
root@asgard:/home/george# ping -f 10.1.1.1
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data.
.............................................
.............................................
CTRL+C
--- 10.1.1.1 ping statistics ---
100 packets transmitted, 3 received, 97% packet loss, time 8148ms
rtt min/avg/max/mdev = 0.240/0.389/2.291/0.450 ms, ipg/ewma 12.053/0.364
ms
```

8.5.2 Traceroute

Utilitarul **`traceroute`** este folosit pentru a testa calea pe care o urmează pachetele între sursă și destinație. El va afișa câte un mesaj pentru fiecare stație/server/echipament de rețea prin care trec mesajele.

Pentru a realiza acest lucru, traceroute se folosește de câmpul **TTL** (*Time To Live*) prezent în antetul fiecărui pachet. Din motive ce țin de felul în care funcționează o rețea este posibil ca în drumul între sursă și destinație să apară bucle. Dacă un pachet intră într-o astfel de buclă, el va fi trimis la infinit între stațiile ce aparțin buclei și nu va ajunge niciodată la destinație. Pentru a preveni o asemenea situație, în antetul fiecărui pachet a fost introdus câmpul TTL. Acest câmp conține numărul maxim de echipamente de rețea prin care poate trece un pachet înainte de a fi aruncat.

Dacă un pachet are câmpul TTL cu valoarea inițială (atunci când pleacă de la sursă) egală cu 10, atunci el poate trece prin maxim 10 echipamente de rețea. Al zecelea

echipament va arunca pachetul și va trimite un mesaj ICMP sursei prin care o va înștiința de acest lucru. Trimițând acest mesaj, al zecelea echipament își va anunța identitatea către sursa pachetului.

Pe acest sistem se bazează funcționarea utilitarului traceroute. Se trimit pachete ce au câmpul TTL configurat la o valoare din ce în ce mai mare. Atunci când valoarea TTL este 1, va răspunde primul echipament de rețea de pe cale și va spune că a aruncat pachetul trimis. Atunci când valoarea TTL este 2, va răspunde al doilea echipament de pe cale și va spune că a aruncat pachetul trimis. În felul aceasta toate echipamentele de pe parcurs vor fi identificate unul câte unul.

Parametrii uzuali ai comenzii **`traceroute`** sunt următorii:

```
traceroute [-f first_ttl] [-m max_ttl] host
```

`host` reprezintă destinația până la care se dorește aflarea rutei:

```
george@asgard:~$ traceroute rol.ro
traceroute to rol.ro (195.95.229.171), 30 hops max, 40 byte packets
 1  10.1.1.1 (10.1.1.1)  0.429 ms  0.332 ms  0.248 ms
 2  10.100.0.1 (10.100.0.1)  0.809 ms  0.657 ms  0.527 ms
 3  backbone.hertza.ro (89.165.146.177)  6.286 ms  1.765 ms  2.493 ms
 4  84.234.111.41 (84.234.111.41)  6.738 ms  7.507 ms  11.869 ms
 5  84.234.107.154 (84.234.107.154)  4.406 ms  1.603 ms  1.625 ms
 6  195.95.229.171 (195.95.229.171)  2.715 ms  1.676 ms  1.230 ms
```

Pentru fiecare echipament de pe parcurs este afișată adresa lui și timpul de răspuns pentru trei încercări succesive. În cazul în care pentru un mesaj trimis nu este primit răspuns, se va afișa caracterul `*`. Parametrul `-f` permite configurarea valorii câmpului TTL din primul mesaj trimis. Valoarea câmpului TTL din mesajele următoare va depinde de aceasta și se va incrementa cu 1 la fiecare mesaj.

```
george@asgard:~$ traceroute -f 4 rol.ro
traceroute to rol.ro (195.95.229.171), 30 hops max, 40 byte packets
 4  84.234.111.41 (84.234.111.41)  2.008 ms  8.286 ms  10.525 ms
 5  84.234.107.154 (84.234.107.154)  3.399 ms  4.614 ms  5.497 ms
 6  195.95.229.171 (195.95.229.171)  4.458 ms  2.901 ms  2.947 ms
```

Parametrul `-m` configurează valoarea maximă a câmpului TTL care va fi încercată în cazul în care destinația nu a fost atinsă. Dacă paramerul nu este specificat, se vor testa implicit căi cu lungimi de până la 30 de echipamente.

```
george@asgard:~$ traceroute -m 3 rol.ro
traceroute to rol.ro (195.95.229.171), 4 hops max, 40 byte packets
 1  10.1.1.1 (10.1.1.1)  0.404 ms  0.402 ms  0.326 ms
 2  10.100.0.1 (10.100.0.1)  0.623 ms  8.407 ms  2.069 ms
 3  backbone.hertza.ro (89.165.146.177)  9.707 ms  6.819 ms  7.895 ms
```

8.5.3 Cum se depistează problemele uzuale în cazul configurărilor de rețea

Atunci când apar probleme în configurările de rețea, pentru depistarea lor există două abordări: de sus în jos (*top-down*) și de jos în sus (*bottom-up*). Partea de sus (*top*) a unei rețele se referă la partea de aplicații, iar partea de jos (*bottom*) se referă la conexiunile fizice între echipamente.

În continuare va fi analizată abordarea de jos în sus, deoarece majoritatea problemelor dintr-o rețea sunt cauzate de conexiunile fizice între echipamente.

Să presupunem că nu poate fi accesată pagina web `www.google.ro`. În cazul acestei abordări primul lucru care trebuie verificat este faptul că există legătură fizică între stație și switch sau punctul de acces fără fir. În cazul legăturilor la Ethernet (realizate prin cablu UTP) conectivitatea se poate verifica examinând led-ul de link de pe placa de rețea. Dacă led-ul este aprins, există conexiune între stație și switch. Dacă led-ul este stins, trebuie verificat cablul de rețea sau funcționarea switch-ului.

O altă metodă folosită pentru a verifica starea link-ului pentru o interfață de rețea, utilă în special atunci când nu există acces fizic la stație, este utilizarea unui program de genul **`ethtool`**. Printre altele, acesta oferă informații detaliate despre tipul de legătură fizică existentă și starea ei. Pentru a vedea starea unei interfețe utilitarul se apelează având ca parametru numele interfeței:

```
root@asgard:/home/george# ethtool eth0
Settings for eth0:
        Supported ports: [ TP MII ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
        Advertised auto-negotiation: Yes
        Speed: 100Mb/s
        Duplex: Full
        Port: MII
        PHYAD: 32
        Transceiver: internal
        Auto-negotiation: on
        Supports Wake-on: pumbg
        Wake-on: d
        Current message level: 0x00000007 (7)
        Link detected: yes
```

În cazul în care `Link detected` are valoarea `yes`, există conexiune între stație și switch. Dacă are valoarea `no`, trebuie verificat cablul de rețea sau funcționarea switch-ului.

Următorul pas este verificarea existenței unui gateway și a conexiunii cu acesta. Pentru a verifica faptul că un gateway a fost configurat trebuie analizat fișierul `/etc/network/ interfaces` sau rezultatul comenzii **`route`**, funcție de configurație. Dacă niciun gateway nu este configurat, trebuie configurat unul. Dacă gateway-ul este configurat, trebuie verificată legătura către acesta cu ajutorul comenzii **`ping`**.

După verificarea gateway-ului, trebuie verificat DNS-ul. Pentru aceasta trebuie analizat fișierul `/etc/resolv.conf`. În cazul în care nu este configurat niciun server DNS, trebuie configurat unul. Dacă este configurat un server, trebuie testată legătura cu acesta cu ajutorul comenzii **`ping`**.

Pasul următor este verificarea conectivității cu un alt server din Internet. Se poate încerca accesarea unei alte pagini web, se poate testa conectivitatea cu un alt server cu ajutorul comenzii **`ping`** sau se poate încerca verificarea căii către `google.ro` cu

ajutorul comenzii **traceroute**. Folosirea **traceroute** oferă avantajul depistării echipamentului de rețea în care se opresc pachetele.

8.6 Studii de caz

8.6.1 Verificarea în linie de comandă a parametrilor de rețea în Windows

Cea mai ușoară metodă de a verifica parametrii de rețea în Windows este folosirea utilitarului **ipconfig**. Acesta se apelează din linie de comandă. Pentru a porni interpretorul de comenzi din Windows XP, se merge la Start→Run, unde în câmpul Open se introduce **cmd** și se apasă pe OK.

Informații despre modul de utilizare al comenzii **ipconfig**, precum și despre parametrii disponibili, se pot obține utilizând parametrul `/?`:

```
C:\Documents and Settings\George >ipconfig /?
```

Apelarea comenzii **ipconfig** fără niciun parametru va afișa parte din parametrii de rețea:

```
C:\Documents and Settings\George>ipconfig
Windows IP Configuration

Ethernet adapter Local Area Connection:
        Media State . . . . . . . . . . . : Media disconnected

Ethernet adapter Wireless Network Connection 2:

        Connection-specific DNS Suffix  . : lan
        IP Address. . . . . . . . . . . . : 192.168.1.70
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.1.254
```

Pentru a vedea toți parametrii de rețea, inclusiv serverul DNS și gateway-ul implicit, se folosește parametrul `/all`:

```
C:\Documents and Settings\George>ipconfig /all
Windows IP Configuration
        Host Name . . . . . . . . . . . . : Asgard
        Primary Dns Suffix  . . . . . . . :
        Node Type . . . . . . . . . . . . : Mixed
        IP Routing Enabled. . . . . . . . : No
        WINS Proxy Enabled. . . . . . . . : No
        DNS Suffix Search List. . . . . . : lan

Ethernet adapter Local Area Connection:
        Media State . . . : Media disconnected
        Description . . . : Broadcom 440x 10/100 Integrated Controller
        Physical Address. : 00-40-F4-CC-C5-41

Ethernet adapter Wireless Network Connection 2:
        Connection-specific DNS Suffix  . : lan
        Description . . . . : Atheros AR5006X Wireless Network Adapter
        Physical Address. . . . . . . . . : 00-19-7E-0B-7E-65
```

```
19          Dhcp Enabled. . . . . . . . . . . : Yes
20          Autoconfiguration Enabled . . . . : Yes
21          IP Address. . . . . . . . . . . . : 192.168.1.70
22          Subnet Mask . . . . . . . . . . . : 255.255.255.0
23          Default Gateway . . . . . . . . . : 192.168.1.254
24          DHCP Server . . . . . . . . . . . : 192.168.1.254
25          DNS Servers . . . . . . . . . . . : 192.168.1.254
26          Lease Obtained. . . . . . . . . . : 11 august 2007 16:17:26
27          Lease Expires . . . . . . . . . . : 12 august 2007 16:17:26
```

Configurarea parametrilor de rețea din line de comandă în Windows este mai dificil de realizat decât în Linux. Pentru a putea face acest lucru se folosește utilitarul **`netsh`**. Informații despre modul de utilizare al acestuia se pot obține folosind parametrul `/?`:

```
1 C:\Documents and Settings\George >netsh /?
```

8.6.2 Configurări de rețea în Windows Vista

Modul de configurare al unei interfețe de rețea în Windows Vista este asemănător cu cel din Windows XP. Diferența importantă între ele este dată de modul în care se accesează fereastra de configurare a parametrilor.

Din meniul Control Panel se alege *Network and Internet* (figura 8.5).

Figura 8.5: Selectarea Network and Internet din Control Panel

Apoi se selectează *Network Sharing Center* (figura 8.6).

Figura 8.6: Selectarea Network Sharing Center

Din noua fereastră se alege *Manage Network Connections* (figura 8.7).

Urmând ca apoi să fie afișată o listă cu interfețele de rețea disponibile. Prin efectuarea unui click dreapta pe simbolul asociat unei interfețe va apărea un meniu contextual asemănător cu cel din Windows XP (figura 8.8).

Prin selectarea opțiunii *Properties* din acest meniu va fi afișată fereastra de configurare a parametrilor interfeței (figura 8.9).

Pentru a configura adresa IP a interfeței, gateway-ul și serverul DNS, se alege itemul *Internet Protocol Version 4*.

Fereastra de configurare a parametrilor este identică celei din Windows XP (figura 8.10).

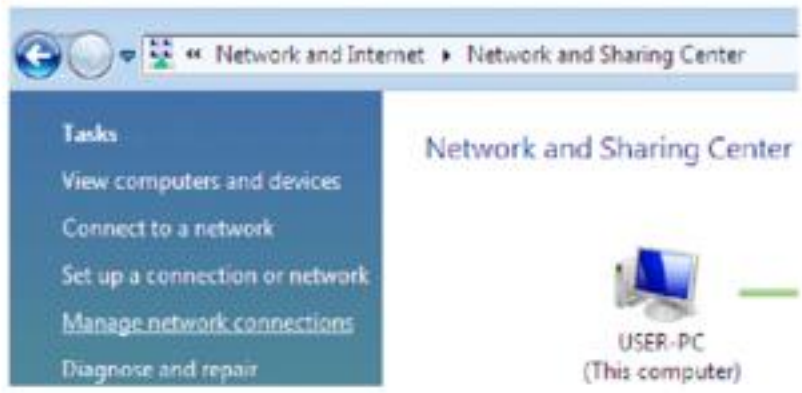


Figura 8.7: Selectarea Manage Network Connections

Figura 8.8: Meniul contextual corespunzător unei interfețe de rețea

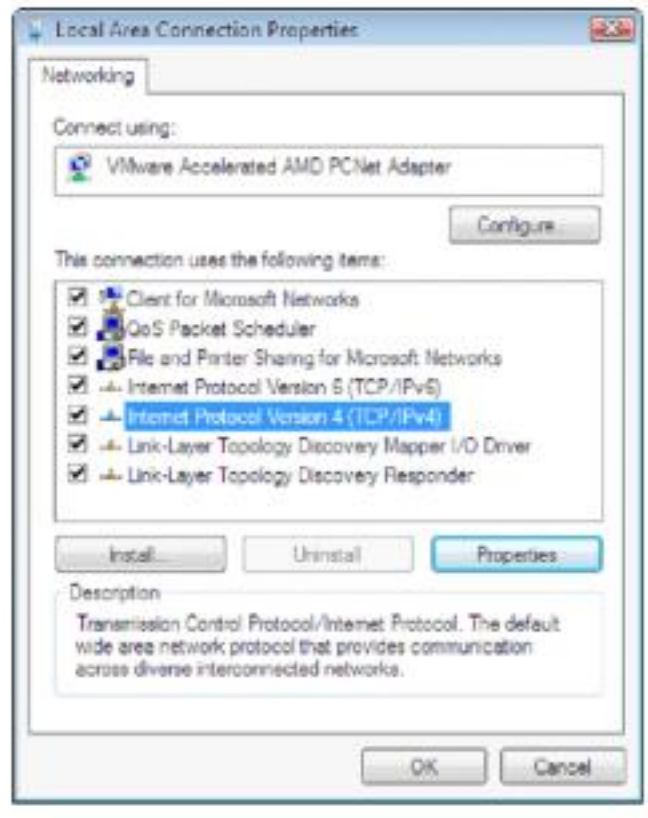


Figura 8.9: Proprietățile unei interfețe de rețea

În cazul în care este aleasă opțiunea *Obtain an IP address automatically*, adresa IP a interfeței va fi configurată prin DHCP. Pentru a specifica manual parametrii interfeței trebuie aleasă opțiunea *Use the following IP address*.

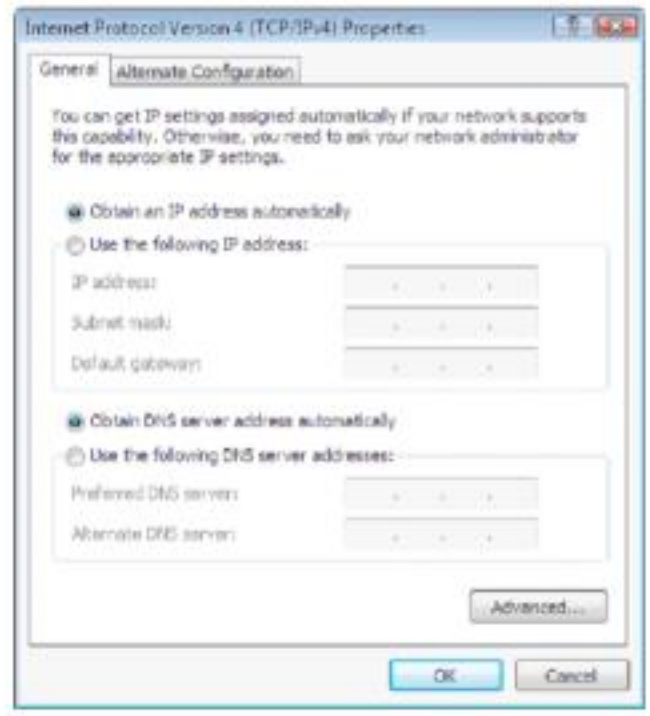


Figura 8.10: Configurarea parametrilor de rețea

8.6.3 Configurarea PPPoE în Linux

PPP (*Point to Point Protocol*) este un protocol utilizat pentru comunicația punct-la-punct (comunicația dintre două noduri conectate direct între ele). PPP permite realizarea de legături peste conexiuni fizice seriale, linii telefonice, conexiuni radio sau fibră optică.

Dintre implementările PPP, cele mai uzuale sunt cele folosite de conexiunile de tip dial-up și PPPoE.

PPPoE (*PPP over Ethernet*) este un protocol care permite încapsularea (includerea) cadrelor PPP în cadre Ethernet, cel mai frecvent pentru a putea fi trimise printr-o placă de rețea obișnuită (ce folosește cablu UTP).

PPPoE este folosit uzual în cadrul serviciilor de tip ADSL sau Internet prin cablu TV. Dacă legătura fizică dintre calculator și modem se realizează prin cablu UTP, atunci este foarte probabil ca legătura logică dintre calculator și modem să folosească PPPoE. Pentru a putea să acceseze Internetul, utilizatorul trebuie mai întâi să stabilească o conexiune PPP cu modemul.

Stabilirea unei conexiuni PPP necesită, de cele mai mule ori, autentificarea utilizatorului. Aceasta se realizează pe baza unui nume de utilizator și a unei parole furnizate de către ISP (*Internet Service Provider* – furnizorul de servicii de Internet).

Pentru configurarea parametrilor conexiunii PPP, inclusiv a parametrilor necesari autentificării, cea mai ușoară metodă este folosirea utilitarului **`pppoeconf`**. Instalarea acestuia depinde de instalarea tuturor pachetelor necesare funcționării protocolului PPP, astfel încât, pe o distribuție ce folosește pachete cum este Ubuntu sau Debian, odată cu **`pppoeconf`** se vor instala și toate dependințele sale.

Pe de altă parte, odată configurată conexiunea PPP cu ajutorul pppoeconf, acesta se va ocupa de realizarea tuturor fișierelor de configurare, inclusiv de adăugarea unui script care la pornirea sistemului de operare va realiza conectarea prin PPP.

Pentru configurarea PPPoE sunt necesare următoarele fișiere (ele sunt create sau actualizate automat de către **`pppoeconf`**):

- `/etc/network/interfaces`
- `/etc/ppp/pap-secrets`
- `/etc/ppp/peers/dslprovider`

Ultimul dintre fișiere poate avea orice nume și se poate afla în orice loc, însă trebuie specificat în cadrul fișierului `/etc/network/interfaces`.

Un exemplu de `/etc/network/interfaces` este următorul:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
iface eth0 inet manual

auto dsl-provider
iface dsl-provider inet ppp
pre-up /sbin/ifconfig eth0 up # line maintained by pppoeconf
provider dsl-provider

auto eth0
```

Autentificarea conexiunii PPP se realizează folosind protocoale specializate. Un exemplu de fișier `/etc/ppp/pap-secrets`, în care sunt trecute informațiile necesare autentificării, este următorul:

```
[...]

# INBOUND connections

# Every regular user can use PPP and has to use passwords from
/etc/passwd
* hostname  ""  *

# UserIDs that cannot use PPP at all. Check your /etc/passwd and add any
# other accounts that should not be able to use pppd!
guest hostname  "*" -
master  hostname  "*" -
root    hostname  "*" -
support hostname  "*" -
stats hostname  "*" -

# OUTBOUND connections

# Here you should add your userid password to connect to your providers
via
# PAP. The * means that the password is to be used for ANY host you
connect
# to. Thus you do not have to worry about the foreign machine name. Just
# replace password with your password.
# If you have different providers with different passwords then you
better
# remove the following line.
# * password

"student" * "usorules"
```

În fișierul `/etc/ppp/peers/dslprovider` sunt trecute informații legate de parametrii conexiunii PPP, alții decât cei de autentificare:

```
# Minimalistic default options file for DSL/PPPoE connections

noipdefault
defaultroute
replacedefaultroute
hide-password
noauth
persist
#mtu 1492
#persist
#holdoff 20
plugin rp-pppoe.so eth0
user "student"
usepeerdns
```

Pentru a activa o conexiune PPP se folosește comanda **`pon`**. Ea este echivalentă cu parametrul up al comenzii **`ifconfig`**. Pentru a dezactiva o conexiune PPP se folosește comanda **`poff`**, echivalentă cu parametrul `down` al comenzii **`ifconfig`**.

Un amănunt important de precizat este că, pentru a funcționa prin PPPoE, o interfață de rețea Ethernet nu necesită configurarea unei adrese IP. Interfața de rețea Ethernet este folosită indirect pentru conectarea la Internet: prin ea sunt trimise datele către modemm acesta asigurând mai departe legătura către Internet.

În acest caz adresa IP este asociată cu o interfața virtuală numită `ppp0`:

```
root@asgard:/home/george# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:14:D1:38:73:A6
          inet6 addr: fe80::240:f4ff:febb:1701/64 Scope:Link
          [...]

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          [...]

ppp0      Link encap:Point-to-Point Protocol
          inet addr:192.168.10.3  P-t-P:10.0.0.1  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1492  Metric:1
          RX packets:176 errors:0 dropped:0 overruns:0 frame:0
          TX packets:180 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:105255 (102.7 KiB)  TX bytes:13991 (13.6 KiB)
```

După cum se poate observa, interfața `eth0` nu are configurată adresa IP. În schimb interfața `ppp0`, asociată conexiunii PPP, este cea care fost configurată cu adresa IP `192.168.10.3`.

Cuvinte cheie

- rețea de calculatoare
- LAN, MAN, WAN
- protocolul IP; adresă IP
- adresă MAC
- Ethernet
- UTP

- ruter
- unicast, multicast, broadcast
- adresă de (sub)rețea
- default gateway
- DNS
- switch
- Access Point
- DHCP
- configurare temporară/permanentă;
- configurare statică/dinamică
- ifconfig
- /etc/network/interfaces
- /etc/resolv.conf
- /etc/hosts
- route
- DHCP
- dhclient
- ping
- traceroute
- ipconfig
- PPPoE

Întrebări

1. Un utilizator încearcă să testeze conectivitatea către `cs.pub.ro` cu ajutorul comenzii **`ping`**: "ping cs.pub.ro" și testul reușește. Totuși, dacă utilizatorul tastează în bara de adrese a browser-ului "cs.pub.ro", pagina web respectivă nu poate fi accesată. Care poate fi motivul pentru care cele două teste au rezultate diferite?
   - ❑ serverul web de la adresa `cs.pub.ro` este oprit
   - ❑ utilizarea comenzii **`ping`** este greșită
   - ❑ comanda **`ping`** a oprit serverul web
   - ❑ utilitarul **`ping`** nu oferă informații despre conexiunea cu `cs.pub.ro`
2. Se poate configura adresa 10.138.257.134 pe interfața de rețea eth0?
   - ❑ da, oricând
   - ❑ da, doar dacă este singura interfața a sistemului
   - ❑ da, doar în situația în care este configurat un server DNS
   - ❑ nu
3. Dându-se o stație care are configurată adresa IP `172.16.150.200` și masca de rețea `255.255.255.0`, care este adresa rețelei din care face parte stația?
   - ❑ `172.0.0.0`
   - ❑ `172.16.0.0`
   - ❑ `172.16.150.0`
   - ❑ `172.16.150.200`
4. Deși legătura la Internet funcționa, utilizatorul Dorel a dorit să testeze conectivitatea cu gateway-ul cu ajutorul comenzii **`ping -f 192.168.10.1`**.

La scurt timp el a fost contactat de administratorul de rețea care l-a rugat să își verifice calculatorul împotriva virușilor. Acest lucru s-a petrecut deoarece:

- ❑ Dorel foloseste Linux și nu Windows, la fel ca majoritatea celorlalți utilizatori.
- ❑ Dorel nu a repornit calculatorul de mult timp.
- ❑ Dorel a generat foarte mult trafic în timp scurt, lucru specific virușilor.
- ❑ Dorel nu obisnuiește să cripteze email-urile trimise.

5. Rolul serviciului DNS este de a:
   - ❑ asocia nume de domenii cu adrese MAC
   - ❑ asocia nume de domenii cu adrese IP
   - ❑ asocia adrese IP cu adrese MAC
   - ❑ asocia adresa IP cu numele utilizatorului
6. În fișierul `/etc/network/interfaces` se găsește următoarea linie:

```
1 iface eth0 inet dhcp
```

   În linia de comandă este dată următoarea comandă:

```
1 ifconfig eth0 192.168.124.150
```

   după care stația este repornită. Care va fi adresa IP a interfeței `eth0` imediat după repornirea stației?
   - ❑ `192.168.124.150`, pentru că a fost ultima adresă configurată de `root`.
   - ❑ `192.168.124.150` dacă aceasta este adresa furnizată stației de serverul DHCP
   - ❑ interfața nu va avea nicio adresă IP pentru că s-a creat un conflict prin configurarea adresei IP în două moduri
   - ❑ interfața nu va avea nicio adresă IP pentru că nu a fost dat parametrul `netmask` comenzii **`ifconfig`**
7. Configurările permanente se realizează cu ajutorul fișierului `/etc/network/ip`. Configurările temporare își pierd efectul o dată cu repornirea sistemului.
   - ❑ adevărat, fals
   - ❑ adevărat, adevărat
   - ❑ fals, fals
   - ❑ fals, adevărat
8. Masca de rețea este utilizată pentru a identifica adresa rețelei din cadrul unei adrese IP. Pentru a putea accesa serverele din Internet este necesară configurarea unui server DNS.
   - ❑ adevărat, fals
   - ❑ adevărat, adevărat

- ❑ fals, fals
- ❑ fals, adevărat

9. Cu ajutorul comenzii **`traceroute`** se poate testa:
   - ❑ Lățimea de bandă a unei conexiuni
   - ❑ Calea dintre sursă și destinație
   - ❑ Funcționarea serverului web de pe stația destinație
   - ❑ Existența unui ruter între sursă și destinație
10. Dacă un server DNS nu este specificat, care dintre următoarele comenzi este probabil să funcționeze:
    - ❑ **`ping www.google.ro`**
    - ❑ **`ping cs.pub.ro`**
    - ❑ **`ping localhost`**
    - ❑ **`ping dns.localdomain.ro`**

Capitolul 9

Servicii de rețea

When I took office, only high energy physicists had ever heard of what is called the World Wide Web.... Now even my cat has its own page.

Bill Clinton, 1996

Ce se învață din acest capitol?

- Conceptul de protocol și stivă de protocoale
- Modelul client-server
- Folosirea porturilor
- Execuția comenzilor la distanță: telnet, ssh, wget, curl
- Funcționarea serviciului de poștă electronică (email)
- Funcționarea serviciului de web
- Modalități de partajare a fișierelor: SMB, FTP

9.1 Concepte specifice aplicațiilor de rețea

9.1.1 Stiva TCP/IP

Sistemele de calcul au început să fie dezvoltate odată cu sfârșitul celui de al doilea război mondial. Cu toate acestea, problematica asigurării comunicației între sisteme aflate în locații la mare distanță a fost formulată abia spre începutul anilor '70, de către US DoD (*US Department of Defense* – Ministerul Apărării Naționale al Statelor Unite). Proiectul US DoD a creat și un cadru de standardizare pentru soluțiile de comunicație, denumit stiva de protocoale TCP/IP (*Transmission Control Protocol/Internet Protocol*).

Un **protocol** este un set de reguli care guvernează modul în care două dispozitive schimbă informație într-o rețea și asigură coerența comunicației.

O **stivă de protocoale** reprezintă un set de protocoale ce comunică între ele prin funcții (numite primitive) clar definite.

O stivă de protocoale este în general privită ca o succesiune de niveluri. Fiecare dintre niveluri este descris de:

- un set de funcții pe care le îndeplinește;
- un set de parametri pe care îi așteaptă de la nivelul inferior;
- serviciile pe care le oferă nivelului imediat superior.

Protocoalele dintr-o stivă nu vor putea comunica între ele direct decât dacă se află pe niveluri adiacente.

Descrierea unei soluții de comunicare prin prisma unei stive de protocoale oferă în primul rând avantajul flexibilității procesului de standardizare. Astfel, schimbarea (sau simpla optimizare) unui protocol din stivă poate fi făcută fără a solicita modificări în restul protocoalelor.

Începuturile rețelelor de date au fost marcate de existența mai multor protocoale de comunicație, dezvoltate în paralel, de vendori diferiți. Problema consta în faptul că, deși două protocoale, de la doi vendori diferiți, descriau amândouă foarte bine specificațiile propriilor dispozitive, nu puteau să comunice unul cu celălalt. Astfel, pentru a face posibilă intercomunicarea între echipamentele de rețea produse de vendori diferiți, s-a introdus conceptul de standardizare a unui protocol. Odată ce un protocol este acceptat ca standard, el devine dintr-un set de reguli, un **set comun de reguli**, fiind recunoscut din punct de vedere procedural și funcțional de toți vendorii.

Pentru a descrie prelucrările efectuate asupra datelor trimise de la utilizator spre mediul de transmisie, stiva TCP/IP identifică patru niveluri, prezentate în figura 9.1: `Aplicație`, `Transport`, `Rețea` (sau `Internet`) și `Acces la mediu`.

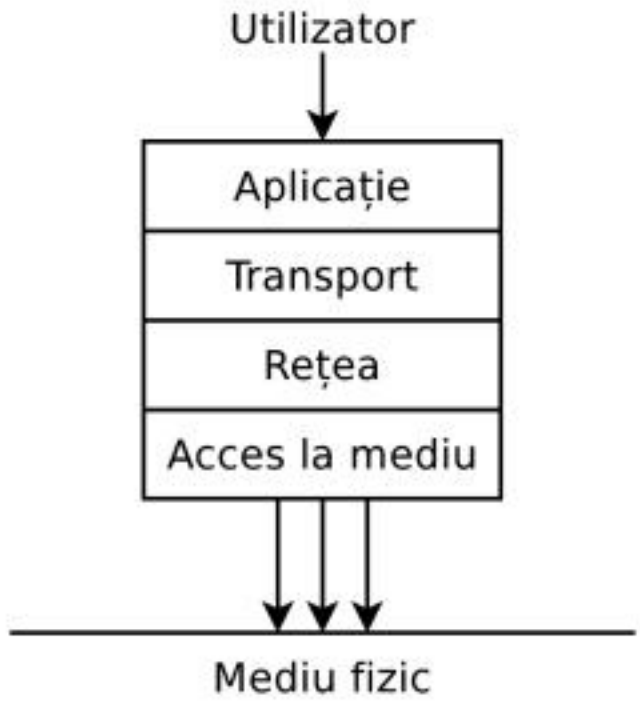


Figura 9.1: Stiva TCP/IP

Nivelul aplicație

Nivelul aplicație este cel mai apropiat de utilizator și oferă acestuia modalitatea de accesare a serviciului de rețea. Un serviciu de rețea poate fi implementat prin unul sau mai multe protocoale de nivel aplicație.

Protocoalele de la acest nivel vor conține reguli referitoare la interacțiunea cu utilizatorul, formatarea și reprezentarea datelor, precum și la controlul dialogului între componentele serviciului.

În funcție de serviciul oferit, protocoalele de nivel aplicație pot fi grupate în:

- serviciul de transfer al fișierelor, ce poate fi implementat folosind FTP (*File Transfer Protocol*), HTTP (*HyperText Transfer Protocol*) etc.
- serviciul de poștă electronică, ce se bazează pe următoarele protocoale: SMTP (*Simple Mail Transfer Protocol*), POP3 (*Post Office Protocol 3*), IMAP (*Internet Mail Access Protocol*)
- serviciul de executarea comenzilor la distanță: **`telnet`**, **`ssh`**, **`rlogin`**
- alte protocoale: DNS (*Domain Name System*), DHCP (*Dynamic Host Configuration Protocol*), SNMP (*Simple Network Management Protocol*)

Nivelul transport

Nivelul transport este responsabil cu probleme legate de siguranță, control al fluxului și corecție de erori. Tot la nivelul transport este realizată multiplexarea comunicației prin specificarea **porturilor sursă și destinație**. Cele mai importante protocoale de nivel transport sunt TCP (*Transmission Control Protocol*) și UDP (*User Datagram Protocol*).

TCP este un **protocol orientat conexiune**, adică înainte de a trimite datele va negocia parametrii de comunicație între entitățile participante la comunicație, iar apoi pe tot parcursul va asigura reordonarea pachetelor ajunse în ordine greșită la destinație, sau retransmisia celor pierdute.

Pentru a înțelege mai bine noțiunea de protocol orientat pe conexiune, se poate face o analogie cu transmiterea datelor prin rețeaua telefonică. În momentul efectuării unei convorbiri telefonice, se stabilește între sursa și destinatie o conexiune dedicată prin rețea, înainte de începerea comunicării. Deasemenea acest exemplu relevă faptul că într-un protocol orientat pe conexiune, destinația este mereu contactată înainte de începerea transmisiei propriu-zise (în analogia cu soneria telefonului).

UDP este un **protocol neorientat conexiune**. Astfel, UDP nu se va preocupa de inițierea sau terminarea conexiunii, de pierderea pachetelor sau doar a succesiunii acestora. UDP va începe transmisia fără a verifica măcar dacă destinația finală va accepta respectivul trafic.

Deși 95% din traficul Internet folosește protocolul TCP, pentru rețelele locale numeroase servicii se bazează pe UDP: DNS, SNMP, TFTP. Motivul principal îl reprezintă gradul redus al erorilor în rețelele locale (spre deosebire de WAN). UDP este mai simplu de implementat, dar și mai eficient decât TCP: antetul (informația suplimentară adăugată de fiecare protocol) UDP este doar de 8 octeți, față de 20 de octeți cât are TCP.

Nivelul rețea

Nivelul rețea (Internet) se ocupă cu transmiterea pachetelor de la o sursă la o destinație precizată, indiferent în ce rețea se află aceasta. Alegerea căii pachetului prin rețelele intermediare, comutarea lui dintr-o o rețea în alta și optimizarea traseelor alese sunt responsabilitatea protocolului de rețea.

Nivelul acces la mediu

Nivelul acces la mediu se ocupă cu toate problemele legate de transmiterea efectivă a unui pachet pe o legătură fizică, incluzând și aspectele legate de tehnologii și de medii de transmisie.

9.1.2 Implementarea parțială a stivei TCP/IP

Dispozitivele de interconectare asigură transferul cât mai rapid al datelor în general pe baza unor algoritmi de decizie. Un dispozitiv de interconectare poate în același timp asigura conversia între două tipuri de rețea (două protocoale) diferite.

Criteriul principal de evaluare a eficienței unui dispozitiv de interconectare este **numărul de pachete comutate pe secundă**. În funcție de complexitatea deciziei de comutare numărul de pachete prelucrate scade. Pentru echipamentele de rețea ce folosesc un număr redus de criterii pentru decizie, se renunță la implementarea nivelurilor superioare din stiva de protocoale TCP/IP.

O primă clasă de dispozitive de interconectare își bazează deciziile de comutare doar pe informațiile protocoalelor de nivel acces la mediu. În această categorie intră **switch-urile**, dar și **AP**-urile (*access point*).

A doua clasă de dispozitive va analiza atât informațiile de nivel acces la mediu, cât și pe cele de nivel rețea, procesul de decizie fiind mai complex, dar și mai lent decât în primul caz. Cel mai important dispozitiv de interconectare de nivel rețea este **ruterul**.

Dispozitivele de interconectare pot implementa doar o parte a stivei de protocoale. Spre exemplu un ruter nu trebuie să înțeleagă conținutul unui pachet de date pentru a putea lua decizia de trimitere pe calea optimă. Astfel, ruterul are nevoie doar de informațiile de nivel acces la mediu și nivel rețea și va implementa doar primele două niveluri ale stivei TCP/IP.

Figura 9.2: Comunicația între două stații aflate în rețele diferite

Considerând o topologie simplă, prezetată în figura 9.2, în care un client de web aflat pe stația B acceseaza un server aflat pe stația A. Cele două stații se află în rețele locale diferite conectate printr-un ruter. Rețeua din stânga este o rețea Ethernet, în vreme ce în rețeua din dreapta AP-ul (acces point) este conectat prin Ethernet cu ruterul R și oferă conectivitate fară fir pentru stația B.

Modelul de comunicație poate fi descris generic folosind stiva de protocoale TCP/IP precum în figura 9.3.

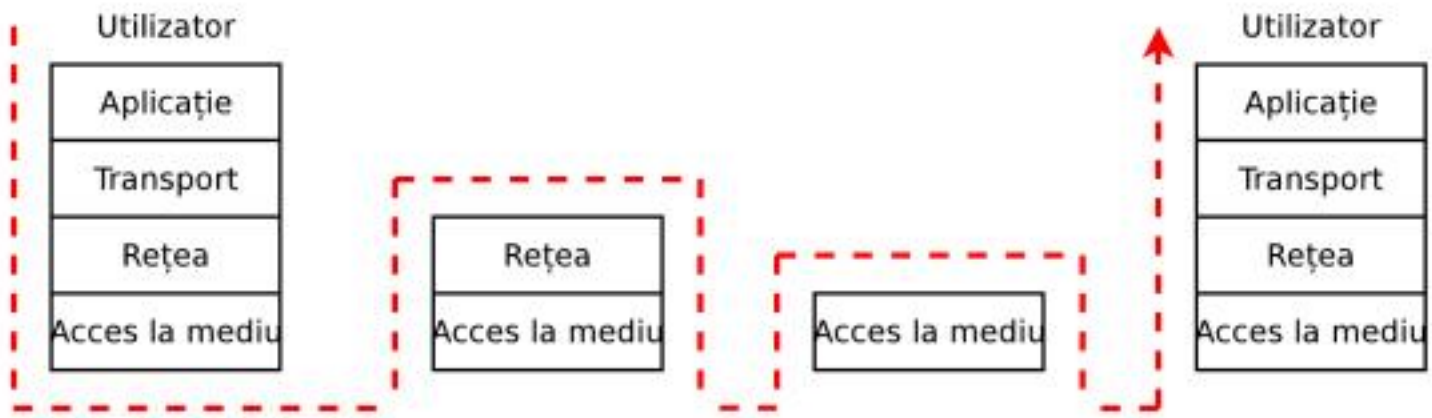


Figura 9.3: Descrierea generică a comunicației

Pentru topologia considerată în figura 9.2, la nivelul aplicație se folosește serviciul de web bazat pe protocolul HTTP. La nivelul transport și nivelul rețea se folosește TCP, respectiv IP. Spre deosebire de nivelul aplicație, unde putem avea o largă diversitate de protocoale folosite, la nivelurile transport și rețea peste 95% din trafic-ul în Internet este trafic TCP/IP.

Pentru nivelul acces la mediu în rețelele locale există două tehnologii dominante: Ethernet (IEEE 802.3) și WLAN (*Wireless* LAN, IEEE 802.11). Conectivitatea între cele două standarde poate fi realizată fie de ruter (cu o interfață 802.11 și una Ethernet), fie de un AP (*acces point*).

Un AP nu are nevoie să interpreteze informația aflată la nivelul rețea pentru a realiza comutarea între rețeaua Ethernet și rețeaua fără fir.

Cele două dispozitive de interconectare (ruterul și AP-ul) au fost figurate ca având câte o stiva parțială de protocoale pe fiecare dintre interfețele de rețea. Stiva interfeței de intrare va fi folosită pentru a lua decizia de comutare a pachetului (alegerea interfeței de ieșire), un echipament de interconectare putând avea mai mult de două interfețe. După determinarea interfeței de ieșire se va folosi stiva acesteia (parametrii specifici acestei interfețe) pentru eventuala reformatare a pachetului.

Pentru exemplul considerat, fluxul de date între stația A și stația B, va solicita funcții din mai multe protocoale, acestea fiind prezentate în figura 9.4.

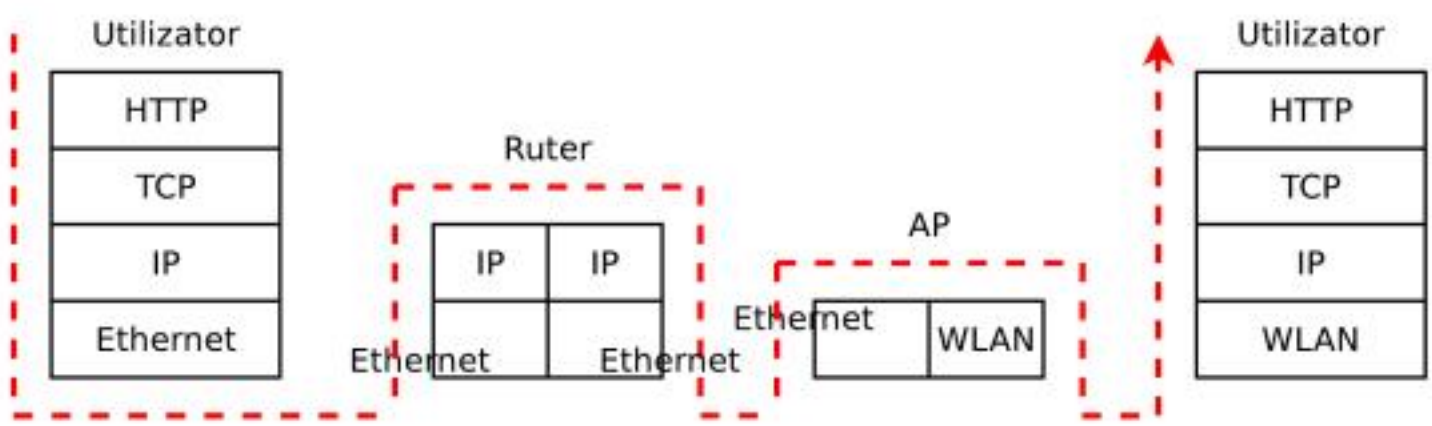


Figura 9.4: Descrierea comunicației între două stații aflate în rețele diferite

9.1.3 Modelul client-server

Pentru implementarea accesului la serviciile de rețea, arhitectura folosită este bazată pe modelul client-server. Acesta se bazează pe definirea a două componente: cea de server și cea de client.

> Un **server** este un program (sau prin extindere o stație) ce permite primirea de cereri de la alte entități din rețea, oferind acestora un serviciu.

> Un **client** este un program ce oferă mecanismul de interogare a unui server și eventual de formatare a răspunsului primit de la acesta.

Un exemplu de **model client server** este serviciul de web. Acest serviciu presupune rularea unui server HTTP (ex: Apache, Microsoft IIS etc). Aplicația client poate fi orice navigator (*broswer*).

Chiar pentru exemplul serviciului de web se poate observa că aplicația client integrează în realitate mai multe aplicații client: pentru HTTP, dar și pentru FTP, TFTP.

Modelul client-server oferă centralizarea serviciilor și a fost în primul rând promovat ca o metodă de reducere a costurilor totale a unui rețele. Ca urmare a scăderii costului resurselor hardware, principală motivație a centralizării serviciilor a devenit monitorizarea și securizarea rețelei.

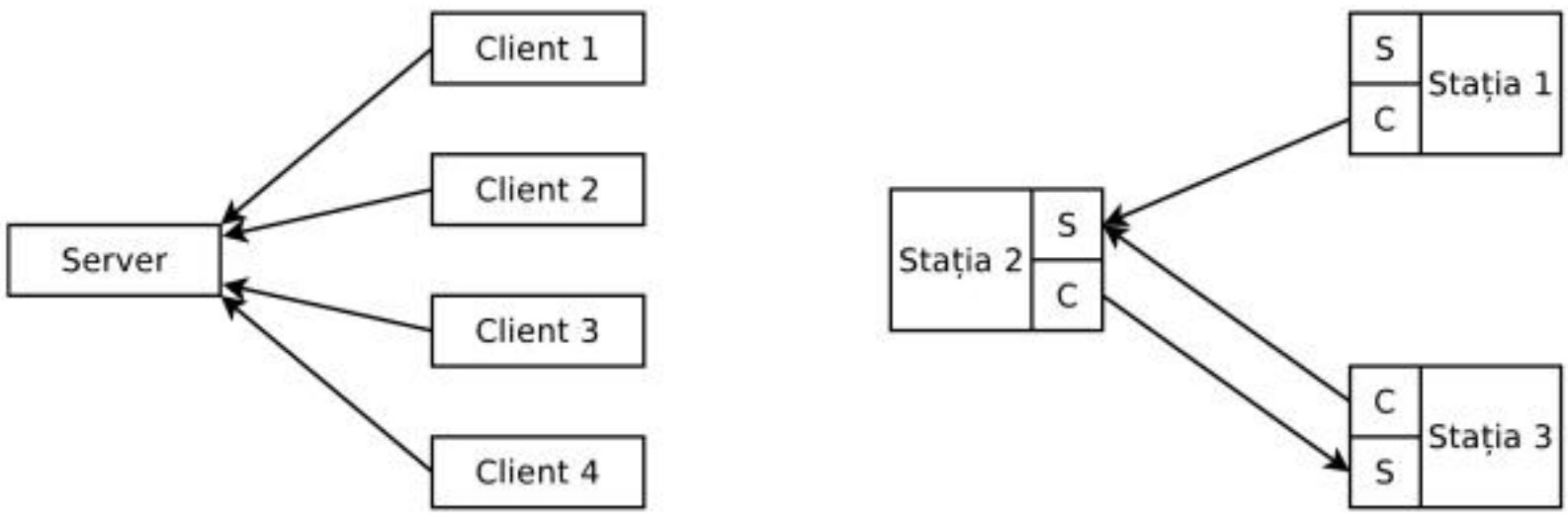


Figura 9.5: Modelul client-server și modelul peer-to-peer

Noțiunea de server s-a extins și pentru sistemele de calcul, specificațiile hardware ale unui server depinzând de serviciul pe care îl îndeplinește. Spre exemplu, un server de stocare va avea un procesor relativ modest, dar un spațiu impresionant de stocare, precum și mecanisme de redundanță a discurilor.

Modelul punct-la-punct (*peer to peer*) este de fapt un model client-server ascuns, așa cum se poate vedea și din figura 9.5. Fiecare entitate ce participă la comunicație poate juca atât rol de client cât și rol de server.

Spre deosebire de modelul client-server, modelul punct-la-punct este un model descentralizat.

Multe din programele de partajare de fișiere utilizează acest model pentru a nu folosi un server care să intermedieze comunicația. Un model client-server de partajare a fișierelor

între clienți presupune ca orice pachet schimbat între cei doi clienți să treacă prin server, ceea ce aduce atât o latență, cât și o încărcare a rețelei și a serverului.

Principalul dezavantaj al centralizării serviciilor este faptul că serverul va fi un punct critic în rețea: în momentul în care serverul nu funcționează, serviciul este întrerupt. Într-o rețea punct-la-punct acest dezavantaj este evitat.

9.1.4 Porturi

În scopul reducerii costurilor unei rețele, mai multe servicii pot fi oferite de aceeași mașină. Spre exemplu, pentru o rețea locală, serverele de DHCP, și DNS pot fi integrate pe o singură stație, deoarece resursele solicitate de fiecare serviciu sunt relativ reduse.

Cum va putea totuși un astfel de server să diferențieze între cererile de rezolvare de nume (cereri DNS) și cele de alocare de adrese IP (cereri DHCP)?

Ambele protocoale folosesc la nivelul transport UDP, deci nu vom putea lua decizia pe baza tipul protocolului de nivel transport.

Atât UDP, cât și TCP folosesc un mecanism de identificare a serviciilor (a protocolului de nivel aplicație) denumit port.

> **Portul** este un număr reprezentat pe 16 biți, ce identifică la nivelul transport (în informația adăugată de protocoalele TCP sau UDP) un serviciu specific.

Mecanismul de porturi permite atât oferirea simultană a mai multor servicii, cât și diferențierea între cereri de răspuns venite din conexiuni diferite, adică folosirea mai multor servicii simultan. Astfel, în antetul de nivel transport vom preciza atât un port destinație, ce identifică serviciul apelat, cât și un port sursă, pe care este așteptat răspunsul. O conexiune va fi descrisă prin precizarea parametrilor de nivel rețea și de nivel transport: adresa IP destinație, adresa IP sursă, port destinație și port sursă.

Dacă de pe aceeași stație sunt lansate două sesiuni de SSH către același server, se creează două conexiuni diferite având aceleași adrese IP (sursă și destinație), același port destinație, dar portul sursă diferit.

Porturile sunt numere reprezentate pe 16 biți, valorile definite pentru porturi fiind cuprinse în intervalul [1, 65535]. Porturile până la 1023 sunt rezervate pentru cele mai importante protocoale ale Internet-ului, fiind denumite porturi rezervate (*known ports*). Porturile rezervate sunt asociate unui protocol pe care serverul îl folosește în oferirea unui serviciu.

Rezumând: fiecare serviciu de rețea este oferit folosind un protocol de nivel aplicație, fiecare protocol are la rândul său asociat un port, în unele cazuri două sau mai multe.

În tabelul 9.1 sunt prezentate câteva dintre protocoalele importante și porturile folosite. Putem observa spre exemplu că serviciul de FTP are rezervate două porturi: 21 (pentru stabilirea conexiunii) și 20 (pentru transferul efectiv de fișiere).

Pentru a determina serviciile oferite pe stația locală va trebui să afișăm lista porturilor deschise (aflate în stare `LISTEN`). Pentru aceasta putem folosi utilitarul **`netsat`** (vezi secțiunea 10.3.1):

Tabelul 9.1: Asocieri port-serviciu pentru câteva servicii răspândite

Port	Serviciu
20	FTP (data)
80	HTTP
21	FTP (conexiune)
110	POP3
22	SSH
143	IMAP
23	telnet
161	SNMP
25	SMTP
443	HTTPS
53	DNS
993	IMAPS

```
root@kiwi~# netstat --tcp --listening
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:sunrpc                *:*                     LISTEN
tcp        0      0 *:www                   *:*                     LISTEN
tcp        0      0 *:webcache              *:*                     LISTEN
tcp        0      0 *:auth                  *:*                     LISTEN
tcp        0      0 *:ftp                   *:*                     LISTEN
tcp        0      0 *:smtp                  *:*                     LISTEN
```

Din rezultatul comenzii de mai sus se poate deduce că portul ftp (adică portul 21) este deschis. Altfel spus, există o aplicație ce folosește acest port pentru a accepta conexiuni. În general, serviciile sunt rulate pe porturile rezervate, dar prin configurarea serviciilor putem schimba portul pe care se vor accepta conexiunile. Pentru a obține mai multe informații despre aplicația ce rulează pe portul 21 putem folosi clientul de **`telnet`** ce va deschide o conexiune pe acest port:

```
razvan@kiwi:~$ telnet kiwi.cs.pub.ro 21
Trying 141.85.99.5...
Connected to kiwi.cs.pub.ro.
Escape character is '^]'.
220 Welcome to Kiwi FTP service. vsFTPd 2.0.3 is in service
```

Dacă oprim serviciul de FTP și reconfigurăm serverul de SSH să primească cereri pe portul 21, rezultatul comenzii **`telnet`** va fi:

```
razvan@kiwi:~$ telnet kiwi.cs.pub.ro 21
Trying 141.85.99.5...
Connected to kiwi.cs.pub.ro.
Escape character is '^]'.
SSH-2.0-OpenSSH_3.8.1p1 Debian-8.sarge.4
```

Din motive de securitate serverele pot fi configurate să nu afișeze informații referitoare la versiunea serviciului, sau să afișeze informații false.

În cazul în care se încearcă inițierea unei sesiuni de telnet pe un port ce nu este deschis va fi întors mesajul de conexiune refuzată:

```
razvan@kiwi:~$ telnet kiwi.cs.pub.ro 29
Trying 141.85.99.5...
telnet: Unable to connect to remote host: Connection refused
```

9.2 Execuția comenzilor la distanță

Serviciul de execuție al comenzilor la distanță (*remote connection*) este oferit de mai multe protocoale, cele mai importante fiind: telnet, rlogin, RSH, SSH.

Protocoalele telnet, rlogin și RSH se bazează pe realizarea unei conexiuni necriptate (în text clar). Spre deosebire de acestea, protocolul SSH stabilește o conexiune criptată, conexiune folosită atât pentru lansarea comenzilor la distanță, cât și pentru transferul de fișiere.

9.2.1 Telnet

În practică, graniță dintre serviciu de rețea și protocol de nivel aplicație este, deseori, una greu de evidențiat. Astfel, putem spune că serviciul de execuție al comenzilor la distanță este oferit de protocolul telnet, dar la fel de corect se poate vorbi de serviciul telnet. Prin serviciul telnet se înțelege posibilitatea execuției comenzilor la distanță peste o conexiune nesigură (necriptată).

Pe lângă denumirea unui protocol de nivel aplicație, și a unui serviciu de rețea, **`telnet`** este și denumirea aplicației ce implementează protocolul (a procesului care din punctul de vedere al sistemului de operare oferă serviciul de rețea).

Aplicația **`telnet`** este o aplicație client-server: procesul server se numește `telenetd`, iar clientul se numește **`telnet`**. Serverul de telnet este responsabil cu acceptarea sesiunilor inițiate de clienți. O sesiune de telnet pune la dispoziția aplicației client (după autentificare) un interpretor de comenzi ce rulează pe server. În mod implicit, nu există diferențe între resursele disponibile unui utilizator în urma rulării unui interpretor de comenzi local, și resursele accesibile printr-o sesiune de telnet de către același utilizator.

```
rookie@apple~$ telnet 141.85.99.5
Trying 141.85.99.5...
Connected to 141.85.99.5.
Escape character is '^]'.
Fedora release 7 (Moonshine)
Kernel 2.6.21-1.3194.fc7 on an i686
login: manager
Password:
manager@kiwi~$
```

În exemplul de mai sus utilizatorul `rookie` inițiază o sesiune telnet către serverul cu adresa 141.85.99.5. După afișarea mesajelor de întâmpinare (denumite MotD – *Message of the Day*) este afișat prompt-ul de login. Este folosit numele de utilizator manager și parola asociată acestui cont. După autentificare va apărea prompt-ul interpretorului de comenzi.

Protocolul telnet a fost standardizat pentru a permite clientului de telnet inițierea de conexiuni la distanță și către alte servere. Majoritatea implementărilor serverelor permit inițierea unei conexiuni prin clientul de telnet (cu unele excepții notabile, precum clienții RDS, cărora le este blocat portul 25 al serviciului SMTP).

Pentru a iniția o conexiune pe serverul de HTTP ce rulează pe stația cu adresa IP 141.85.99.5 trebuie rulată aplicația client **`telnet`** cu precizarea destinației și a portului pe care rulează serviciu (implicit 80). După stabilirea sesiunii se pot executa inclusiv comenzi ce respectă sintaxa protocolului HTTP, spre exemplu pentru a obține o pagină se folosește comanda HTTP GET, urmată de pagina ce se dorește afișată.

```
1  razvan@kiwi:~$ telnet 141.85.99.5 80
2  Trying 127.0.0.1...
3  Connected to localhost.localdomain.
4  Escape character is '^]'.
5  GET /~razvan/test.html HTTP/1.0
6
7  HTTP/1.1 200 OK
8  Date: Fri, 03 Nov 2008 19:57:10 GMT
9  Server: Apache/1.3.33 (Debian GNU/Linux) PHP/4.3.10-16
10 Last-Modified: Fri, 03 Nov 2008 19:56:40 GMT
11 ETag: "253095-44-454b9ef8"
12 Accept-Ranges: bytes
13 Content-Length: 68
14 Connection: close
15 Content-Type: text/html; charset=iso-8859-1
16
17 <br />Pagina de test pentru cursul USO - Servicii de Retea <br />
18 Connection closed by foreign host.
```

Este important de reținut că pentru terminarea unei sesiuni de telnet se poate folosi combinația `CRTL+]`.

9.2.2 SSH

SSH (*Secure Shell*) este un protocol de nivel aplicație, folosit în rețele de calculatoare pentru asigurarea unui canal securizat de transmitere a informației.

Protocolul funcționează după modelul client-server. Pentru a iniția o conexiune, clientul de SSH trimite o cerere către serverul de SSH, iar acesta din urmă, după autentificarea clientului, permite transferul de date.

Cea mai întalnită implementare a protocolului SSH 2.0 este varianta open-source numită OpenSSH.

Protocolul este adesea folosit pentru executarea comenzilor la distanță, dar poate fi deasemenea utilizat și pentru transfer securizat de fișiere sau tunelarea altor protocoale mai nesigure (FTP). Există multe alte protocoale de nivel aplicație, mai vechi, cu care se pot obține aceleași efecte precum cu SSH: FTP, rlogin, rsh, telnet. Toate aceste protocoale au însă în comun aceeași slăbiciune: folosesc metode de autentificare slabe, ce presupun trimiterea parolelor pe rețea în format text clar (*clear text*). Formatul text clar reprezintă un risc maxim de securitate, capturarea parolelor făcându-se relativ simplu de o persoană cu intenții malițioase.

Pentru asigurarea securității, SSH folosește algoritmul RSA de generare de **chei publice și private**. După cum denotă și numele celor două chei, cheia privată trebuie păstrată întotdeauna secretă, iar cea publică poate fi cunoscută de oricine. Există o strânsă relație matematică între cele două chei, care face imposibilă derivarea cheii private din cea publică, dar care permite decriptarea unui mesaj criptat cu una dintre chei, folosind cealaltă cheie. Dacă se dorește trimiterea unui mesaj privat către destinatarul X, tot ce trebuie făcut este criptarea mesajului cu cheia publică a destinatarului (fiind publică, toată lumea are acces la această cheie). Odată criptat, acest mesaj poate fi decriptat doar de destinatarul X, căci el este singurul care deține cheia privată corespunzătoare cheii publice cu care s-a făcut criptarea.

Utilizarea SSH pentru efectuarea conectării la distanță

Comanda de conectare la un server SSH are doi parametri importanți: numele utilizatorului și adresa (numele) serverului destinație. Serverul de SSH la care se va face conectarea (`securessh.pub.ro` în exemplul de mai jos), este un nume public pe care serviciul de DNS îl va traduce într-o adresă IP. Dacă se dorește conectarea la un server dar nu se cunoaște numele său DNS, se poate introduce direct IP-ul serverului. Când un client inițiază o conexiune ssh pe un server, trebuie să se conecteze folosind un utilizator existent pe acel server pentru a putea avea acces la interpretorul de comenzi. Parametrul `bogdand` specifică utilizatorul în contul căruia se va intra după stabilirea conexiunii.

```
1 razvan@kiwi:~$ ssh bogdand@securessh.pub.ro
2 Password:
3 Last login: Wed Sep 19 14:37:29 2007 from 86.121.138.243
4 Welcome to the dark side.. we've got cookies!
5
6 bogdand@securessh:~$
```

În rezultatul comenzii de mai sus se poate observa faptul ca s-au mai făcut conexiuni anterioare pe acel server de SSH, fiind oferită atât ora și data ultimei conexiuni, cât și adresa IP de la care s-a realizat conexiunea.

Când un client de ssh inițiază pentru prima dată o conexiune către un server nou, acesta din urmă va trebui să se autentifice către client. Fiecare server de SSH are un identificator unic (*host key*) cu care se autentifică în comunicația cu clienții. Acest identificator este implementat prin chei publice și chei private. La primirea unei conexiuni de la un client, serverul oferă cheia sa publică, făcând astfel posibilă autentificarea. La acceptarea cheii publice de la server, clientul adaugă această cheie în fișierul `known_hosts`. Astfel, următoarea conexiune pe care acest client o va face către server nu va mai necesita transferul cheii publice, aceasta existând deja stocată pe client.

```
1 razvan@kiwi:~$ ssh bogdand@82.77.5.245
2 The authenticity of host '82.77.5.245 (82.77.5.245)' can't be established
  .
3 RSA key fingerprint is 11:94:89:c3:e5:fe:c3:6c:2e:ab:f6:e2:bd:6d:bb:e7.
4 Are you sure you want to continue connecting (yes/no)? yes
5 Warning: Permanently added '82.77.5.245' (RSA) to the list of known hosts
  .
6 Password:
```

```
Have a lot of fun...

bogdand@server:~$
```

Un utilitar pentru sistemele Windows ce permite conectarea în linie de comandă la un server de SSH este aplicația PuTTY. PuTTY este disponibil gratuit și este un client pentru protocolul SSH, telnet, rlogin, dar poate fi folosit și în comunicațiile *TCP raw*. O privire de ansamblu asupra opțiunilor oferite de clienții importanți de SSH poate fi găsită online[1].

Utilizarea SSH pentru rularea de comenzi la distanță

Sintaxa completă a utilitarul **ssh** permite precizarea unei liste de parametri, a utilizatorului și a adresei destinație, precum și a unei comenzi ce va fi rulată după stabilirea sesiunii SSH. Dacă nu este precizată o comandă se va rula un interpretor de comenzi (cel mai adesea `/bin/bash`).

În exemplul de mai jos sunt rulate local două comenzi separate prin ";". Apoi se rulează aceleași comenzi (de data aceasta protejate între ghilimele), rezultatul afișat fiind cel al executării lor pe stația destinație, după autentificarea cu utilizatorul `bogdand`. Se observă din exemplu că autentificarea se realizează fără a interoga utilizatorul, aceasta fiind rezultatul unei autentificări pe bază de chei (vezi secțiunea 10.5.2):

```
rrazvan@apple:~$ hostname; pwd
apple
/home/rrazvan

rrazvan@apple:~$ ssh bogdand@141.85.99.5 "hostname;  pwd"
kiwi
/home/users/bogdand

rrazvan@apple:~$
```

Copierea fișierelor la distanță

Efectuarea transferului de date se poate face prin utilitarul **scp**, ce face parte din pachetul `ssh`. Ideea din spatele **scp** este de a folosi canalul securizat oferit de **ssh** pentru a face transfer de date în format text sau binar între client și server. Sintaxa **scp** este asemănătoare comenzii tradiționale **cp**:

```
scp sursa destinatie
```

Dacă se dorește transferul mai multor fișiere sau directoare, destinația trebuie să fie un director existent. Pentru a efectua o copiere de fișier de pe client, în directorul `/home/bogdand/work/` de pe serverul cu adresa IP 82.77.5.245, comanda este:

```
razvan@kiwi:~$ scp CiscoTFTP.exe bogdand@82.77.5.245:~/work
Password:
CiscoTFTP.exe                                 100% 1296KB 129.6KB/s   00:10

razvan@kiwi:~$
```

[1] http://en.wikipedia.org/wiki/Comparison_of_SSH_clients

Pentru platforma Windows o aplicație *open source* ce poate fi folosită pentru partajarea de fișiere peste o conexiune SSH este WinSCP. Aceasta oferă o interfață grafică pentru transferul de fișiere prin SCP, SFTP sau FTP.

Serverul de SSH

Pentru a putea permite conexiuni SSH spre stația locală, trebuie instalat serverul de OpenSSH (pentru Ubuntu pachetul se numește `openssh-server`). Fișierul de configurare pentru serverul de SSH se află în `/etc/ssh/sshd_config`, și conține directive standard precum: portul pe care se acceptă conexiuni SSH sau opțiunea de a putea rula aplicații X peste conexiunea criptată de SSH.

```
1  razvan@kiwi:/etc/ssh$ cat sshd_config
2  [...]
3   Port 22
4   X11Forwarding yes
5  [...]
```

Autentificarea cu chei

Pentru o securitate sporită a utilizării SSH, se poate folosi autentificarea cu chei publice și chei private. În cazul standard al folosirii SSH, fiecare stație păstrează o asociere între utilizator și parolă în mod automat. Dacă se dorește însă utilizarea criptării bazată pe chei, devine obligația utilizatorului de a crea această asociere manual. Pentru a genera o pereche de chei se poate folosi utilitarul **`ssh-keygen`** instalat odată cu pachetul OpenSSH.

```
1  razvan@kiwi:~$ ssh-keygen -t rsa
2  Generating public/private rsa key pair.
3  Enter file in which to save the key (/home/waters/.ssh/id_rsa):
4  Enter passphrase (empty for no passphrase):
5  Enter same passphrase again:
6  Your identification has been saved in /home/waters/.ssh/id_rsa.
7  Your public key has been saved in /home/waters/.ssh/id_rsa.pub.
8  The key fingerprint is:
9  c1:45:40:92:b6:f6:e5:0b:72:23:73:ff:88:58:27:d0 waters@waters-laptop
10
11 razvan@kiwi:~$
```

Odată generate, cele două chei sunt stocate în mod implicit în două fișiere: `id_rsa` (cheia privată) și `id_rsa.pub` (cheia publică). Cheia publică trebuie să ajungă pe stația unde se află serverul de SSH, iar cheia privată va rămâne local pe stația unde au fost generate cheile.

În momentul creării cheilor, utilizatorului i se va cere un șir de caractere (*passphrase*) care este asociat cu perechea de chei, și care va fi cerut ulterior, în locul parolei. Acest șir poate fi vid, caz în care nu se va mai cere intervenția utilizatorului în cazul utilizării cheilor pentru autentificare. Folosirea unui *passphrase* vid devine deosebit de utilă pentru automatizarea unor acțiuni (prin scripturi).

Pentru a copia cheia publică pe stația aflată la distanță se poate folosi un utilitar dedicat, **`ssh-copy-id`**, specificând ca parametru adresa serverului SSH.

```
1 razvan@kiwi:~$ ssh-copy-id bogdand@82.77.5.245
```

La următoarea încercare de conectare prin SSH, singura diferență va fi cerea *passphrase*-ul (dacă a fost introdus la generarea cheilor) în locul unei parole de cont, însă, din punct de vedere al securității, utilizarea serviciului va fi mai sigură.

9.2.3 Utilitarul wget

wget este un utilitar gratuit (din suita GNU Project), creat pentru a facilita o descarcare simplă și eficientă a resurselor de pe Web. Aceste resurse pot fi obținute cu **wget** prin două dintre cele mai folosite protocoale din Internet: HTTP și FTP. Numele utilitarului este obținut de fapt printr-o alăturare semantică a World Wide Web (W) și a metodei `HTTP GET`.

Principalul avantaj al **wget** este faptul că execută o **descărcare neinteractivă** a fișierelor. Într-o aplicație din mediul grafic (Firefox, Filezilla), o descărcarea uzuală necesită prezența pe sistem a utilizatorului. Cu alte cuvinte, acesta trebuie să fie logat pentru a se putea efectua operația de descărcare, sau pentru a avea posibilitatea de a reactiva o descărcarea întreruptă de către server.

În contrast, folosind **wget**, utilizatorul poate să se autentifice pe sistem, să pornească descărcarea, și să se deautentifice, **wget** continuând să ruleze până la terminarea sarcinilor neîncheiate. Mai mult, dacă conexiunea către server este întreruptă, utilitarul va stoca fișierele descărcate până în acel moment pe hard-disc, va salva parametrii sesiunii și adresa serverului și va putea continua sarcina de descărcare în momentul în care legătura cu serverul se reface și resursa este din nou accesibilă.

Prezentând un astfel de avantaj evident față de clienții de descărcare din mediul grafic, **wget** a fost rapid adoptat de marea majoritate a utilizatorilor care aveau o conexiune înceată la Internet și care nu puteau petrece autentificați pe sistem timpul necesar unei descărcări interactive. Pe măsură ce Web-ul a evoluat **wget** adăugat noi funcționalități.

Un avantaj notabil al utilitarului **wget** este capabilitatea de descărcare și de păstrare pe client a structurii de directoare de pe server, fiind de asemenea posibilă descărcarea unui site cu indicarea nivelului de adâncime până la care să fie descărcată structura de directoare.

Acest tip de descărcare, numit **descărcare recursivă**, a fost extins până la posibilitatea de a descărca pe client site-uri întregi cu toate paginile și legăturile dintre acestea. Considerând situația unei rețele locale, descărcarea unui site pe mașina server din rețea (proces numit și oglindirea site-ului), ar face implicit posibilă accesarea acelui site în mod *offline* de către toți clienții din rețeaua locală.

Modul de apelare al utilitarului în linia de comandă

Pentru a descărca un fișier prin HTTP se poate folosi comanda:

```
1 razvan@kiwi:~$ wget -c http://ccnp.catc.ro/resources/CiscoTFTP.exe \&
2
3 --15:57:06--  http://ccnp.catc.ro/resources/CiscoTFTP.exe
```

```
4             => 'CiscoTFTP.exe'
5
6  Resolving ccnp.catc.ro... 141.85.37.2
7  Connecting to ccnp.catc.ro|141.85.37.2|:80... connected.
8  HTTP request sent, awaiting response... 200 OK
9  Length: 1,327,497 (1.3M) [application/x-msdos-program]
10
11 100%[====================================>] 1,327,497      2.66M/s
12
13 15:57:07 (2.66 MB/s) - 'CiscoTFTP.exe' saved [1327497/1327497]
```

Opțiunea `-c` asigură continuarea descărcării în cazul întreruperii conexiunii cu serverul, iar simbolul `&` specifică rularea utilitarului în *background*.

Descărcarea unui site pe discul local pentru a face posibilă accesarea sa *offline* este exemplificată prin comanda următoare:

```
1  razvan@kiwi:~$ wget --convert-links -r http://www.gnu.org/ -o gnulog
2
3  razvan@kiwi:~$ ls
4  gnulog  www.fsf.org  www.gnu.org
```

Parametrul `--convert-links` transformă legăturile (linkurile) din site pentru ca site-ul să poată fi vizionat *offline*.

Descărcarea tuturor legăturilor de pe siteul `www.yahoo.com` cu o adâncime de 1 se face prin:

```
1  razvan@kiwi:~$ wget -r -l1 www.yahoo.com
```

9.3 Email – Poșta electronică

> **Poșta electronică** (electronic mail), abreviat e-mail sau email, este o metodă de compunere, transmitere și recepție a mesajelor peste sisteme de comunicație electronică.

Ideea de mesagerie electronică datează din anul 1971, când Ray Tomlinson dezvolta prima aplicație de email pentru ARPANET. Aceasta era formată din două programe: SNDMSG, utilizat pentru a transmite mesaje, respectiv READMAIL, folosit pentru citirea mesajelor. În anul 1972, comenzile `MAIL` și `MLFL` au fost adăugate programului FTP. Aceasta a fost modalitatea de transmitere a mesajelor în rețeaua ARPANET până la începutul anilor '80, când a fost dezvoltat protocolul SMTP. În scurt timp, serviciul de email a crescut popularitatea ARPANET-ului și a devenit aplicația principală a acestuia.

Serviciul de email a oferit la început doar transferul de mesaje text între utilizatori aflați în rețele diferite. Ulterior serviciul a fost extins și la transmiterea de fișiere binare. Standardul de codificare a informației binare într-un mesaj e-mail se numește MIME (*Multipurpose Internet Mail Extensions*).

9.3.1 Arhitectură și funcționare

Pentru livrarea mesajelor se folosește un mecanism de adresare ce apelează atât ierarhia de nume din Internet (serviciul de DNS) pentru a preciza serverul, cât și informații relevante local, precum numele utilizatorului.

O adresă de email conține două tipuri de informații separate prin simbolul @ (citit la sau *at* sau *a-round*):

- numele utilizatorului care dorește transmiterea sau recepția mesajului;
- numele (DNS) al server-ului care transmite/recepționează mesajul.

Pe baza adresei de email destinație serviciul de poștă electronică livrează mesajele de la clienții de email, folosind servere de email, către căsuța poștală asociată cu adresa de email destinație.

> **Căsuță poștală** este o intrare în sistemul local de fișiere utilizată pentru stocarea mesajelor de poștă electronică corespunzătoare unui utilizator.

Componenta esențială a serviciului de poștă electronică o reprezintă protocolul **SMTP** (*Simple Mail Transfer Protocol*). Acest protocol oferă mecanisme pentru preluarea mesajelor de la clienți de email și livrarea lor pe serverele de email destinație (ca atare pentru trimiterea mesajelor).

Odată cu dezvoltarea sistemelor de operare pentru calculatoarele personale, a apărut și nevoia preluării noilor mesaje din căsuța poștală pe stația utilizatorului. Acest lucru este realizat printr-un nou serviciu ce asigură ridicarea email-ului și sincronizarea căsuței poștale de pe server cu cea de pe stația de lucru.

Pentru ridicarea mesajelor de pe serverul de email cele mai folosite protocoale sunt **IMAP** (*Internet Message Access Protocol*) și **POP3** (*Post Office Protocol version 3*). Pentru oferirea acestui serviciu va trebui pornit un daemon de IMAP sau POP3, în plus față de serverul de SMTP.

Marea majoritate a clienților de email integrează pe lângă componenta de client SMTP (folosită pentru transmiterea mesajelor) și componente de client IMAP și POP3 pentru interogarea căsuței poștale de pe serverul de email.

O deosebire importantă a IMAP față de POP3 o reprezintă extinderea posibilităților de prelucrare a mesajelor din căsuța poștală oferite de protocolul IMAP. Folosind IMAP mesajele nu sunt descărcate de pe server decât la vizualizarea lor, existând totodată și opțiunea descărcării doar a unei părți din mesaj.

Spre deosebire de POP3, unde toate mesajele se găsesc pe server într-o singură căsuță poștală (*inbox*), iar directoarele nu se pot crea decât în aplicația client, protocolul IMAP permite crearea de directoaret pe server direc. Utilizând comenzi IMAP, aplicația client poate folosi filtre pentru mutarea mesajelor dintr-un director în altul, fără să fie nevoie măcar ca *mailbox*-urile să fie situate pe același server.

Rezumând, arhitectura serviciului de email include două componente principale: un serviciu pentru primirea și trimiterea mesajelor și un serviciu pentru ridicarea mesajelor de pe serverul de email.

Trimiterea unui mesaj

Pentru a exemplifica modul în care componentele prezente mai sus interacționează, vom considera următoarea situație: Dan are adresa de mail dan@a.org și dorește să-i transmită un mesaj lui Bogdan care are adresa de mail bogdan@b.org.

Pentru livrarea unui mesaj în căsuța poștală asociată contului bogdan@b.org, pașii sunt prezentați în figura 9.6.

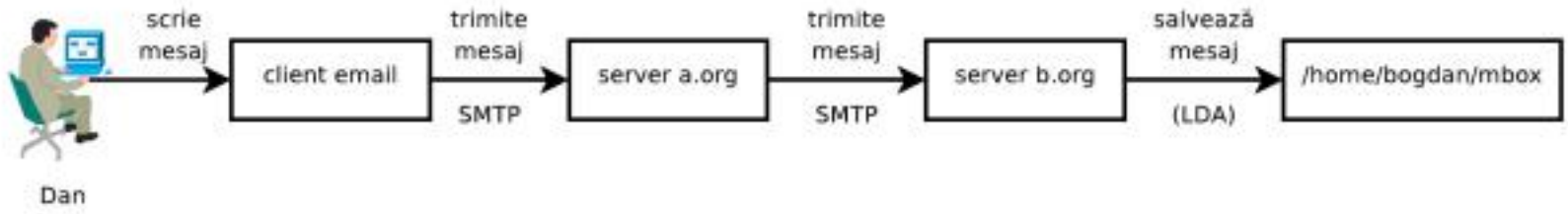


Figura 9.6: Livrarea unui mesaj

Dan va deschide un client de email în care va edita mesajul. Când Dan va da comanda de trimitere a mesajului, clientul de email va iniția o conexiune SMTP către serverul de mail local. Pentru stabilirea conexiunii, serverul va cere autentificarea utilizatorului Dan. În general, clientul de mail va salva numele de utilizator și parola și va realiza autentificarea conexiunii SMTP în mod transparent pentru utilizator. După autentificare, clientul de email va formata mesajul conform protocolului SMTP și îl va trimite către server.

Serverul de mail local va analiza apoi adresa de email destinație, pe baza căreia va determina domeniul de nume în care trebuie livrat mesajul (în cazul exemplului: b.org). Va iniția o cere de rezolvare de nume (cerere DNS) pentru a determina adresa serverului responsabil cu preluarea mailului pentru domeniul b.org (acest server se numește *mail exchanger*). După determinarea serverului de mail pentru b.org, serverul local va trimite mesajul folosind protocolul SMTP.

Serverul b.org va stoca mesajul în căsuța poștală a utilizatorului `bogdan`. La nivelul acestui pas se poate interpune o aplicație de distribuție a mesajelor denumită generic LDA (*Local Delivery Agent*). O astfel de aplicație poate realiza filtrarea mesajelor, livrarea anumitor mesaje în alte directoare, oprirea spamurilor etc. În Linux cea mai folosită aplicație de distribuție a mesajelor este oferită de pachetul `procmail`.

Ridicarea mesajelor

Pentru a ridica mesajul din căsuța poștală vor fi urmați pașii din figura 9.7:

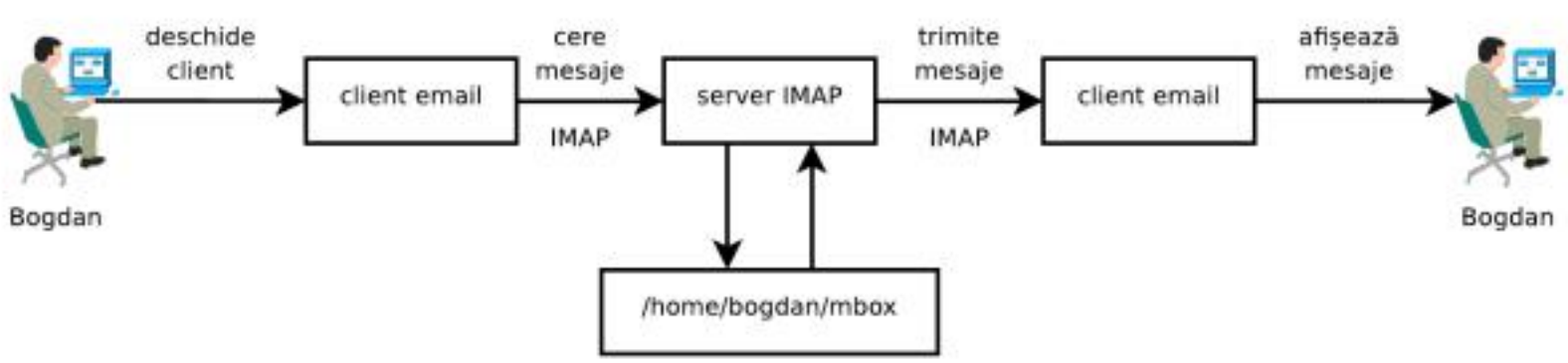


Figura 9.7: Ridicarea mesajelor de pe server

Bogdan va deschide clientul de mail, iar acesta va iniția o sesiune de IMAP către serverul de mail. După autentificare, serverul va deschide căsuța poștală a utilizatorului Bogdan. Mesajele ridicate din căsuța poștală în sesiunile anterioare de IMAP au fost marcate, astfel serverul va selecta doar noile mesaje (ce nu sunt marcate) și le va transmite prin sesiunea de IMAP către clientul de email. Înainte de a încheia sesiunea, clientul de email va informa serverul asupra listei mesajelor ce au fost șterse din căsuța poștală aflată pe stația utilizatorului, permițând astfel serverului să sincronizeze conținutul căsuței poștale locale cu cea aflată pe stație.

După ridicarea mesajelor de pe server, căsuța poștală poate fi consultată inclusiv *offline*, folosind clientul de email.

Folosirea webmail pentru citirea și transmiterea mesajelor

O altă metodă disponibilă utilizatorilor pentru citirea și trimiterea mesajelor este folosirea unui server de web pentru interfața cu serverul de SMTP, serviciu rezultat fiind denumit **webmail**.

Folosind exemplul anterior, mesajul livrat prin SMTP a fost stocat în căsuța poștală a utilizatorului Bogdan. Utilizatorul va lansa un client web (un broswer) și va iniția o conexiune HTTP către serverul de web ce rulează pe același sistem cu serverul de SMTP, responsabil cu primirea mesajelor pentru domeniul b.org. Serverul de web va oferi o interfață ce permite operații direct pe căsuța poștală aflată pe server: ștergerea/mutarea mesajelor, marcarea lor, crearea de directoare etc.

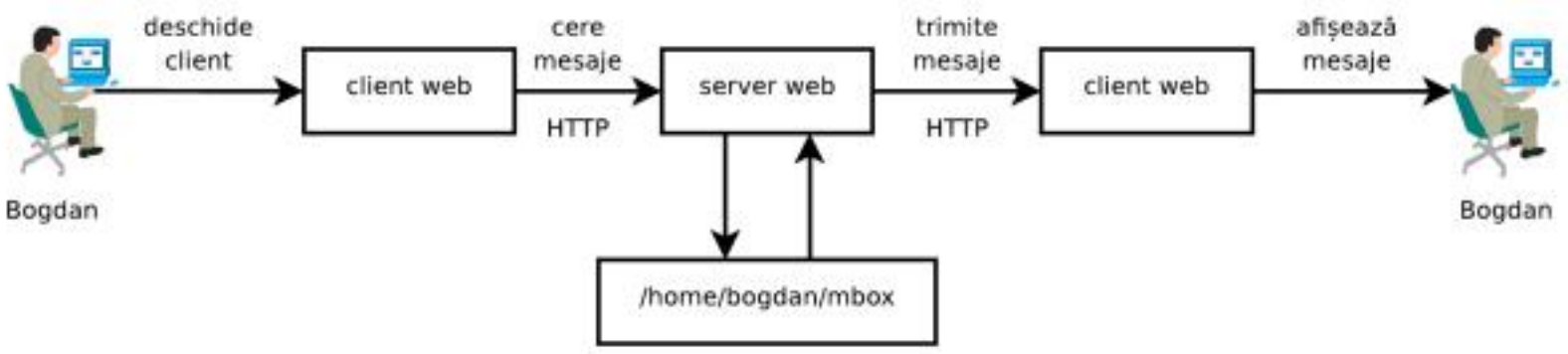


Figura 9.8: Citirea mesajelor folosind webmail

9.3.2 Clienți de email

Așa cum s-a prezentat mai sus, clienții de email sunt utilitare care permit descărcarea de mesaje de pe un server de email și stocarea lor local. În mod real, există două situații în care este necesară folosirea lor: atunci când contul de email nu poate fi vizualizat printr-o interfață de tip web, sau atunci când conexiunea la Internet este slabă sau instabilă, și este de preferat salvarea corespondenței pe computerul personal pentru a fi ușor accesată și ulterior. Există mai mulți clienți de email, cei mai importanți fiind Microsoft Outlook (varianta inclusă în distribuția de Windows se numește Outlook Express) și Mozilla Thunderbird.

Un avantaj al clienților de email este stocarea centralizată a mesajelor primite pe conturi de email diferite. În cele ce urmează, ne vom concentra asupra Mozilla Thunderbird,

deoarece este gratuit și disponibil pentru toate sistemele majore de operare (Windows, Linux, Mac OS).

Configurarea unui client de email presupune în primul rând crearea unui cont pentru fiecare adresă de email ce se dorește administrata cu Thunderbird. Sunt folosite două protocoale diferite: SMTP pentru a trimite mesaje și POP3 pentru a primi mesaje. Setarea unui cont presupune introducerea adresei de email, parolei asociate, adresa serverului de SMTP și a serverului de POP3. Se poate opta pentru salvarea parolei într-un fișier de configurare, sau pentru introducerea ei manuală de fiecare data când se va stabili o conexiune la unul dintre serverele de mail. În cazul în care doar unul dintre cele două servere este setat corespunzător, se vor putea doar descărca (în cazul POP3) sau trimite mesaje (respectiv SMTP).

Unul dintre avantajele Thunderbird este integrarea cu serviciul de email oferit de Google, numit Gmail. Pentru setarea unui cont de Gmail, trebuie parcurși următorii pași: se activează pe site-ul Gmail opțiunea de a folosi un server de POP3, apoi din Thunderbird se creează un nou cont, se alege *Gmail account* și se introduce adresa de email. Restul setărilor se efectuează automat.

Pentru a trimite/primi corespondență, se folosește butonul *Get Mail*, comandă ce sincronizează baza de date locală cu cea de pe server. În consecință, un email care este scris poate fi trimis instantaneu sau poate fi stocat local și trimis mai târziu, când va exista conectivitate la Internet. Thunderbird se sincronizează în mod automat cu serverul la intervale regulate de timp.

Thunderbird oferă o gamă de servicii standard: memorarea adreselor de email, sortarea emailurilor după, data primirii, subiect, numele expeditorului, importanță (crescător sau descrescător). Emailurile pot fi grupate pe threaduri (toate mesajele care fac parte din aceeași discuție), facilitate deosebit de folositoare în cazul mesajelor primite de la o listă de discuții. De asemenea pentru localizarea unui mesaj clientul oferă un motor de căutare ce poate folosii criterii ce vor analiza: adresa sursă, adresa destinație, subiectul mesajului sau conținutul său.

Pentru o mai buna organizare a corespondenței, se poate crea o structură de directoare și gruparea mesajelor în mod logic, în opoziție cu stocarea tradițională a tuturor email-urilor în căsuța poștală (*inbox*). Aplicarea de filtre permite selectarea emailurilor pe bază de criterii avansate (ce includ adresă, nume expeditor, subiect, conținut) și efectuarea de acțiuni asupra lor (mutarea în diferite directoare, ștergerea lor, marcarea lor), facilitate extrem de utilă în cazul unui număr mare de mesaje zilnice.

O limitare în folosirea unui soluții mixte bazate pe Thunderbird și Gmail o reprezintă nesincronizarea marcării mesajelor citite între cele două aplicații.

9.3.3 Securitatea serviciului de email

Cea mai întâlnită problemă asociată cu poșta electronică o reprezintă primirea de spam (mesaje nesolicitate). Din păcate, traficul de **spam** a depășit 90% din traficul SMTP în Internet.

Pentru a reduce traficul de spam în primul rând trebui ca serverul de email (SMTP) să nu accepte să trimită mesaje de la orice client SMTP din Internet. Majoritatea destinațiilor

de mail vor verifica pentru fiecare mesaj dacă nu a fost trimis folosind un server SMTP de tip *open mail relay* (server ce acceptă să trimită mesaje pentru orice client), caz în care vor considera respectivul mesaj drept spam.

Traficul de spam poate fi controlat și pe serverul destinație înainte de a fi preluat de aplicația client. Reducerea traficului de spam pe serverul de SMTP poate fi făcută prin implementarea a trei tipuri de liste:

- **Liste albe** – mesajele trimise de un server conținut într-o astfel de listă sunt considerate automat mesaje legitime
- **Liste gri** – mesajele primite de la servere necunoscute sunt, pentru o perioadă de timp, respinse. Dacă serverul sursă încercă retransmiterea mesajului și după expirarea acestei perioade, mesajul va fi considerat legitim. Această metodă se bazează pe faptul că serverele de spam nu vor încerca retransmiterea mesajelor.
- **Liste negre** – mesajele primite de la un astfel de server sunt etichetate ca spam

O altă clasă de aplicații de filtrare a traficului de spam sunt integrate în LDA (*Local Delivery Agent*). Aceste aplicații în general folosesc un set extins de criterii pentru a calcula probabilitatea ca un mail să fie spam.

Filtrarea traficului de spam poate fi realizată inclusiv de clientul de mail. Cu toate acestea, chiar și în cazul folosirii simultane a tuturor metodelor de control al traficului nedorit, mesajele de spam nu pot fi total eliminate.

O altă problemă importantă a serviciului de email este **absența unui mecanism de autentificare a expeditorului**. Există extensii ale protocolului SMTP ce rezolvă această problema, dar care nu se bucură de o popularitate însemnată. Singura soluție cu adevărat eficientă în prevenirea atacurilor bazate pe asumarea unui false identități (numite atacuri phising) este instruirea utilizatorilor asupra limitărilor protocolului SMTP.

9.4 WWW

World Wide Web sau WWW este, probabil, cel mai cunoscut și utilizat serviciu din Internet. WWW-ul se prezintă ca un set de resurse interconectate prin intermediul de legături (*hyperlink*-uri). Aceste resurse sunt denumite pagini web și conțin *hypertext*, imagini, filme. *Hypertext*-ul este în esență text simplu, la care s-a adăugat o formatare standardizată, ce specifică utilizarea de *hyperlink*-uri. Accesul la resurse se realizează prin intermediul unui navigator (*browser*). O pagină web va conține, de obicei, legături (*hyperlink*-uri) către alte pagini (resurse), justificând denumirea de Web.

World Wide Web-ul este invenția lui **Sir Tim Berners-Lee** din 1989, una din personalitățile marcante în dezvoltarea Internet-ului. Sir Tim Berners-Lee este directorul World Wide Web Consortium (W3C), organizația care coordonează dezvoltarea WWW.

În ultimii ani, WWW-ul a cunoscut o dezvoltare mult mai amplă ajungându-se la folosirea expresiei **Web 2.0** pentru a simboliza serviciile web de nouă generație oferite de diversele comunități web. În același timp, WWW-ul evoluează către ceea ce Sir Tim Berners Lee a denumit **Semantic Web**, adică o extensie a WWW-ului în care conținutul poate fi exprimat în limbaj natural și utilizat de agenți specifici de analiză.

9.4.1 Tehnologiile de bază ale WWW

WWW-ul își datorează existență celor trei tehnologii utilizate de Sir Tim Berners-Lee. Acestea sunt URL, HTTP și HTML și sunt prezentate în continuare.

URL

URL înseamnă *Uniform Resource Locator* și este un nume într-o sintaxă bine precizată pentru a identifica resursele din Internet. URL este de fapt un sinonim pentru URI (*Uniform Resource Identifier*).

În mod tipic, un URL/URI este introdus în bara de adrese a unui navigator și este folosit pentru accesarea unei resurse (a unei pagini web) din Internet. Un navigator va "decodifica" URL-ul și va accesa resursa dorită. Astfel, în cazul în care se accesează URL-ul `http://www.example.com` se face o cerere HTTP către server-ul `example.com`. Pe de altă parte, URI-ul `mailto:bogdan@example.com` va însemna deschiderea unui client de e-mail pentru compunerea unui mesaj către bogdan@example.com.

Un URL are o sintaxă bine specificată folosită pentru identificarea stației din Internet, a serviciului folosit de aceasta și a resursei pe care o poate pune la dispoziție acest serviciu. Astfel, în cazul URL-ului `http://www.example.com/page.html` stația care deține resursa este server-ul `www.example.com`. O cerere DNS va fi folosită pentru translatarea numelui într-o adresă IP. Șirul `http://` este folosit pentru identificarea serviciului; în cazul de față acesta este serviciul de Web care folosește protocolul HTTP și portul implicit `80`. Resursa cerută de pe server este `/page.html`. Drept urmare, în urma introducerii URL-ui în bara de adrese a navigatorului, acesta va face o cerere HTTP către stația identificată de numele `www.example.com` pentru pagina `/page.html` de pe server. Aceasta va fi afișată apoi clientului (în interiorul *browser*-ului).

Exemplul de URL prezentat mai sus este unul simplificat. Astfel, sintaxa completă pentru un URL este:

```
1 protocol://server:port/cale/catre/resursa
```

- câmpul `protocol` specifică, evident, protocolul folosit pentru comunicație. Acesta poate fi `ftp`, `http`, `scp` etc.; dacă acest câmp lipsește, un browser va folosi implicit `http`;
- câmpul `server` reprezintă numele sau adresa IP a stației cu care se dorește realizarea unei comunicații;
- câmpul `port` este portul utilizat (acesta poate lipsi), caz în care se folosește portul implicit al serviciului;
- `/cale/catre/resursa` reprezintă calea către resursa aflată pe server; acesta va folosit această cale pentru a oferi respectiva resursă clientului; de obicei, resursa este o intrare în sistemul de fișiere al server-ului.

Sintaxa URL-ului poate fi complicată. Astfel, un URL tip ftp poate asigura inclusiv autentificarea pe server. Folosirea URL-ului `ftp://bogdan:p4r0l48un4@example.com/sources.tgz` înseamnă

autentificarea utilizatorului `bogdan` cu parola `p4r0l48un4` pe serverul `example.com`, folosind `ftp` pentru descărcarea fișierului `sources.tgz`.

HTTP

HTTP (*Hypertext Transfer Protocol*) este protocolul fundamental al World Wide Web-ului. Deși navigatoarele pot folosi alte protocoale în afara HTTP, acesta rămâne cel mai important mecanism de comunicare.

Similar celorlalte servicii din Internet, serviciul Web (HTTP) funcționează ca sistem client-server. Clientul Web este navigatorul (browser-ul) folosit pentru cererea unei resurse prin intermediul unui URL. Portul implicit utilizat de HTTP este 80.

Protocolul HTTP este un protocol de tip întrebare răspuns (*request – response*). Clientul Web (navigatorul) face o cerere către un server, după care acesta îi trimite un răspuns și resursa cerută. În exemplul de mai jos, telnet este utilizat pentru a efectua o cerere HTTP către `google.ro`:

```
1 razvan@anaconda:~$ telnet google.ro 80
2 Trying 72.14.221.104...
3 Connected to google.ro.
4 Escape character is '^]'.
5 GET / HTTP/1.0
6 HTTP/1.0 302 Found
7 Location: http://www.google.ro/
8 Cache-Control: private
9 [...]
```

După realizarea conexiunii pe portul 80, clientul trimite cererea `GET / HTTP/1.0`. Aceasta înseamnă citirea directorului rădăcină al serverului folosind HTTP versiunea 1.0. Răspunsul server-ului este `HTTP/1.0 302` Found urmat de pagina efectivă. Dacă pagina dorită nu se află pe server, răspunsul server-ului va fi `Error 402 Page not Found`. Acest răspuns nu înseamnă însă că serverul nu funcționează, caz în care mesajul afișat de client ar fi `Server not Found`.

O altă caracteristică importantă a protocolului HTTP este faptul că este un protocol neorientat pe conexiune (*stateless*). Acest lucru înseamnă că după ce clientul trimite o cerere și îi sosește răspunsul conexiunea este încheiată. Pentru o nouă resursă trebuie realizată o altă conexiune. Opus este protocolul FTP care este un protocol cu conexiune persistentă. Petru că este nevoie ca anumite informații să fie persistente de-a lungul mai multor conexiuni s-au găsit diverse soluții: cookie-uri pe client, sesiuni pe server, variabile HTTP etc.

Un alt neajuns al HTTP este lipsa unui suport puternic de securitate. Acest lucru este corectat prin folosirea protocolului **HTTPS** care folosește comunicație criptată. Majoritatea sistemelor de webmail de astăzi folosesc HTTPS pentru a asigura confidențialitatea datelor.

HTML

HTML (*Hypertext Markup Language*) este un limbaj folosit pentru a descrie conținutul unei pagini Web. HTML folosește **marcaje** (tag-uri) pentru a defini modul în care un

element dintr-o pagină Web este afișat de un browser.

O pagină simplă Web descrisă în format HTML este următoarea:

```
razvan@anaconda:~/public_html$ cat main.html
<html>
        <head>
                <title>Pagina mea</title>
        </head>

        <body>
                <h1>Antet</h1>

                Text simplu
                <strong>Text aldin</strong>
                <em>Text cursiv</em>
        </body>
</html>
```

Un marcaj încadrează de obicei un text. La început, marcajul este `<nume_marcaj>` și se încheie cu `</nume_marcaj>`. Marcajele prezente în această pagină sunt:

- **html**, pentru definirea documentului HTML
- **head** prezintă informații despre pagină, spre exemplu titlul (marcajul title)
- **h1** definește un antet de nivel 1
- **strong** definește un text aldin (bold)
- **em** definește un text cursiv (italic)

Un client Web va interpreta o pagină HTML și o va afișa utilizatorui conform marcajelor existente.

XHTML

XHTML (Extensible Hypertext Markup Language) este un standard și un limbaj de marcare care are aceleași posibilitați de exprimare ca și HTML însă se conformează sintaxei mult mai stricte a XML. Urmând o sintaxă mult mai strictă, un document XHTML este mult mai ușor de interpretat, spre deosebire de un document HTML care necesită un parser complex.

Motivația utilizării XHTML o reprezintă creșterea numărului de dispozitive care nu au capacitatea de procesare a unui document HTML complex. Sintaxa strictă a XHTML permite o prelucrare mult mai rapidă a unei pagini Web de către client.

Adoptarea XHTML se realizează într-un ritm inegal datorită lipsei de implementare în navigatoare a caracteristicilor noi pe care standardul le aduce.

Versiunea curentă de XHTML este 1.1. Versiunea 2.0 a standardului este încă în dezvoltare.

9.4.2 Funcționarea serviciului

După cum s-a precizat, serviciul de Web funcționează în sistem client-server și folosește protocolul HTTP pentru comunicare. Funcționarea serviciului este sistematizată în figura 9.9, în care sunt prezentați un set de pași urmați tipic în obținerea de către client, a unei resurse aflate pe un server Web (de obicei o pagină Web).

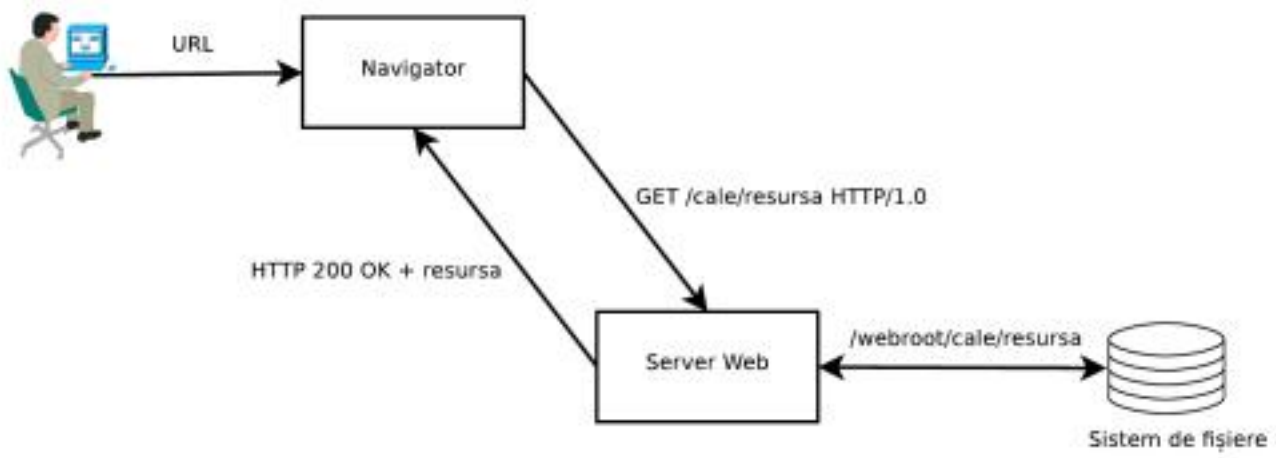


Figura 9.9: Funcționarea serviciului de Web

În primă fază utilizatorul va folosi un client Web (navigator) și va introduce în bara de adrese a acestuia un URL care identifică resursa dorită.

Navigatorul identifică din URL sistemul unde trebuie să se conecteze și resursa cerută. De obicei se va genera și o cerere DNS pentru a afla adresa IP asociată serverului. Navigatorul se conectează la server și, în cazul protocolului HTTP, formulează o cerere de forma `GET /cale/resursa HTTP/1.0`.

Cererea ajunge la serverul Web. Acesta trebuie să ofere clientului resursa cerută. Resursa este, de obicei, o intrare în sistemul de fișiere (pagină web, imagine, film etc.). Drept urmare serverul Web va trebui să localizeze resursa. Orice server web are configurat un director rădăcină specializat, denumit și **webroot**. Orice resursă este obținută din acest director. Drept urmare, calea completă în sistemul de fișiere este obținută prin concatenarea căii din cererea HTTP la *webroot*, rezultând intrarea `/webroot/cale/ resursa`.

După ce obține resursa obținută, serverul va trebui să o transmită clientului. Pentru aceasta el emite un răspuns `HTTP 200 OK` și atașează resursa.

Clientul primește resursa de la server și, dacă este vorba de o pagină Web, interpretează codul (X)HTML și afișează rezultatul utilizatorului. Conexiunea se închide.

9.4.3 Servere Web

Un **server Web** este responsabil cu oferirea de resurse din sistemul local de fișiere către clienți (navigatoare) prin intermediul protocolului HTTP. Exemple de servere Web sunt Apache HTTP Server, Microsoft IIS, Sun Web Server, Zeus Web Server. Apache HTTP Server este cel mai folosit server Web din Internet.

În general, un server Web prezintă un fișier de configurare prin intermediul căruia îi poate fi alterat comportamentul. În cazul Apache HTTP Server versiunea 2, acest fișier

de configurare este `/etc/apache2/apache2.conf`. Alterarea acestui fișier de configurare permite schimbarea portului pe care acesta ascultă conexiuni (implicit 80), a webroot-ului (implicit `/var/www`) și altele.

Un server Web va primi o cerere HTTP în forma `GET /cale/catre/resursa HTTP/ 1.0` și va identifica din sistemul local de fișiere intrarea asociată resursei. Fiecare server are un director rădăcină (denumit *webroot*) unde este căutată resursa pentru a fi oferită clientului. În cazul serverului Apache, în urma unei cereri `GET /cale/catre/resursa HTTP/1.0` va trimite clientului fișierul identificat de `/var/www/cale/catre/resursa` (sau `/usr/local/Apache2/htdocs`).

9.4.4 Clienții Web

Clienții Web se numesc **navigatoare** sau **browser**-e. Un client Web se va conecta la un server Web pentru a solicita o resursă. Resursa este specificată de utilizator într-un URL. Clientul Web transmite o cerere HTTP către server solicitând resursa clientului. De obicei această resursă este o pagină Web. Codul HTML este interpretat apoi de navigator și afișat utilizatorului.

Cele mai utilizate tipuri de navigatoare sunt Internet Explorer, Mozilla Firefox, Opera și Safari. Internet Explorer este cel mai utilizat navigator cu o pondere de 80-85% urmat de Mozilla Firefox cu o pondere de 10-15%. Celelalte navigatoare au ponderi mult mai mici. Există și navigatoare în linie de comandă, cum ar fi **`lynx`**, **`w3m`**, **`links`**.

Navigatoarele Web reprezintă una dintre cele mai utilizate aplicații din cadrul unui sistem de operare. Competiția dintre acestea a generat ceea ce s-a numit "*browser wars*". Astfel, anii 1994-1998 au fost marcați de competiția dintre Netscape Navigator și Internet Explorer cu victoria celui din urmă. În anul 2002, Internet Explorer era folosit în proporție de circa 92%. Ultimii ani sunt dominați de competiția între Mozilla Firefox și Internet Explorer.

Competiția dintre navigatoare a condus și la apariția unui număr important de caracteristici printre care: navigare tabulară, blocarea mesajelor de tip pop-up, corecție gramaticală, gesturi de mouse, suport pentru diverse protocoale (FTP, IRC, BitTorrent) și pentru standarde (CSS, XHTML, XSLT, RSS).

9.5 Studii de caz

9.5.1 Utilitarul cURL

Utilitarul cURL este aplicație pentru transferul de fișiere ce își propune să ofere suport atât pentru cerințe avansate de securitate, cât și pentru un număr foarte mare de protocoale. Lista protocoalelor implementate de cURL include FTP, HTTP, telnet, LDAP, precum și protocoale orientate spre securitate precum SCP, SFTP, FTPS, HTTPS. Pentru asigurarea securității se pot folosi în cURL numeroase metode de autentificare pe bază de parolă, cât și pe certificate.

Utilitarul cURL ce se execută din linia de comandă, ceea ce permite includerea sa în scripturi. Cel mai adesea cURL este folosit pentru scripting HTTP, în scopul simulării activității utilizatorilor ce folosesc diferite broswere.

Utilitarul cURL implementează numeroase mecanisme specifice clienților web, cele mai multe suplimentare față de **wget**. Spre exemplu oferă gestionarea *cookies* (trimiterea și salvarea locală), navigarea prin pagini de formular, folosirea unui proxy web, urmărirea redirectărilor automate, sau schimbarea semnăturii navigatorului.

Spre exemplu, presupunem că în cadrul unui site se află un formular disponibil la adresa: `test.cs.pub.ro/login.cgi`. În acest formular trebuie completat un câmp de nume și apoi apăsat butonul OK pentru a accesa site-ul. Formularul este scris folosind sintaxa HTML:

```
1 <form method="GET" action="login.cgi">
2     <input type=text name="nume">
3     <input type=submit name=press value="OK">
4 </form>
```

Listing 9.1: Formular de login simplu

Cererea poate fi trimisă folosind cURL astfel:

```
1 razvan@kiwi:~$ curl "test.cs.pub.ro/login.cgi?nume=Razvan\&press=OK"
```

În cazul unui formular ce permite clientului să încarce un fișier se poate folosi cURL cu opțiunea `-F`. Presupunem că formularul este scris astfel:

```
1 <form method="POST" enctype='multipart/form-data' action="upload.cgi">
2     <input type=file name=upload>
3     <input type=submit name=press value="OK">
4 </form>
```

Listing 9.2: Formular de upload simplu

Pentru formularul aflat la adresa `test.cs.pub.ro/upload.cgi` va fi încărcat fișierul `test01` și apoi simulată apăsarea butonului OK:

```
1 razvan@kiwi:~$ curl -F upload=@test01 -F press=OK test.cs.pub.ro/upload.
  cgi
```

Utilitarul cURL oferă suport și pentru schimbarea identității navigatorului. Pentru a simula o cerere către serverul test.com inițiată de un navigator Internet Explorer 7.0 ce rulează pe un sistem Windows 2000 vom folosi comanda:

```
1 razvan@kiwi:~$ curl -A "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT
  5.0)" test.co
```

9.5.2 FTP

FTP (*File Transfer Protocol*) este un protocol foarte răspândit ce oferă posibilitatea transferului de fișiere atât într-o rețea locală cât și prin Internet. Se bazează pe modelul client-server, deci presupune existența a două entități între care se realizează un transfer de fișiere, clientul fiind cel care se conectează (și se autentifică) la serverul de la care poate descărca sau căruia îi poate trimite fișiere.

Ca și în cazul modelului generalizat de client-server, serviciul FTP oferit de către acesta din urmă se reduce la existența unei aplicații, numită "server FTP" ce rulează pe o stație. Similar, clientul rulează o aplicație numită "client FTP" ce se ocupă de managementul conexiunii către server și de traficul de date dintre aceștia. Atât aplicațiile server cât și cele client FTP sunt disponibile, practic, pentru toate platformele existente, dovadă a versatilității și răspândirii protocolului FTP.

Un dezavantaj al protocolului FTP este faptul că acesta transferă atât informațiile de conectare cât și conținutul propriu-zis al fișierelor în mod necriptat, interceptarea unui astfel de trafic putând duce la o interpretare facilă a sa.

Funcționare

Un client FTP ce se conectează la un server trimite acestuia un set de date de autentificare (nume și parolă) care au asociate pe server un o structură de directoare ce sunt puse la dispoziția acelui utilizator. Dacă autentificarea reușește, clientul are dreptul să trimită în continuare comenzi pentru a interacționa cu fișierele la care are acces. Drepturile de acces la fișiere se reduc la cele de listare/creare de directoare, citire, modificare, ștergere de fișiere sau directoare, ceea ce implică un transfer bidirecțional între client și server (clientul poate atât să descarce, cât și să trimită fișiere pe server). În practică, aceste drepturi sunt enunțate în aceeași structură de tip "rwx" ca și pe sistemele UNIX.

Serverul FTP "ascultă" conexiunile pe portul 21 și realizează transferul de comenzi tot pe acesta. Transferul efectiv de fișiere se realizează în paralel, pe portul 20, doar în urma unei conexiuni prestabilite. Porturile rezervate pentru protocolul FTP sunt 21 pentru stabilirea conexiunii și 20 pentru transferul efectiv de date. O variantă securizată și similară o reprezintă SFTP (SSH File Transfer Protocol). O altă limitare a sa este imposibilitatea de a transmite atributele ce țin de data și ora creării/modificării fișierelor, acestea fiind rescrise la destinație. De asemenea, această limitare este eliminată în cazul lui SFTP.

Transferul datelor prin FTP se poate face prin două moduri, selectate manual înaintea inițierii transferului sau determinate automat de către clientul FTP:

- Mod ASCII: folosit pentru transferul fișierelor text; are avantajul de a transmite datele folosind codurile ASCII ale caracterelor, ceea ce permite compresia dinamică a informației, la un raport foarte bun, deci un flux mai puternic de date, dar are dezavantajul de a corupe conținutul oricărui fișier non-text.
- Mod binar: folosit pentru transferul oricărui tip de fișiere (inclusiv text)

Exemplu de utilizare

Conectarea la un server FTP dintr-o linie de comandă Linux se poate face direct prin comanda **`ftp`**, urmată de numele sau adresa serverului la care se dorește conectarea, ca în exemplul:

```
1 rookie@localhost~$ ftp ftp.lug.ro
2 Trying 193.226.140.51...
3 Connected to ftp.lug.ro (193.226.140.51).
```

```
220 (vsFTPd 2.0.3)
Name (ftp.lug.ro:root): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Promptul `ftp>` indică faptul că următoarele comenzi vor fi trimise direct către server. Câteva comenzi comune sunt ls sau dir pentru a lista conținutul directorului curent, cd pentru a schimba directorul, get pentru a descărca fișierele specificate ca parametri (`mget` pentru fișiere multiple) sau put pentru trimiterea de fișiere. Deconectarea de la serverul FTP se face prin comenzile quit sau disconnect.

O categorie specială de clienți FTP o reprezintă broswerele web, limitarea acestora constând în faptul că nu permit decât descărcarea fișierelor de pe server. Conectarea la un server FTP printr-un browser web se poate face prin introducerea adresei sale în bara de adrese a browserului, specificându-se protocolul:

```
ftp://ftp.lug.ro
```

În cazul serverelor în care accesul nu se poate face anonim (fără cont predefinit), browserul va cere un nume de utilizator și o parolă.

Pentru a specifica direct din adresa parametrii de autentificare, se poate scrie:

```
ftp://user:parola@server:2555
```

Exemplul anterior specifică explicit și portul pe care se realizează conexiunea (2555), informație necesară în cazul în care serverul nu rulează pe portul implicit, 21.

Cuvinte cheie

- client
- curl
- FTP
- HTML
- HTTP
- IMAP
- IP
- LDA
- POP3
- port
- punct-la-punct
- scp
- server
- SMB
- spam
- SSH (Secure Shell)
- stivă de protocoale
- TCP
- telnet
- UDP
- URL
- webmail
- wget
- WWW
- XHTML

Întrebări

1. Care dintre operațiile de mai jos NU se bazează pe o arhitectură client-server?
   - ❑ ridicarea mesajelor
   - ❑ editarea unui nou mesaj
   - ❑ trimiterea unui nou mesaj
   - ❑ verificarea mesajelor folosind broswer
2. Protocolul telnet comunică direct (fără intermedierea nivelului transport) cu nivelul rețea. După stabilirea unei conexiuni de telnet comunicația se va face doar unidirecțional: de la server către client.
   - ❑ adevărat, adevărat
   - ❑ adevărat, fals
   - ❑ fals, adevărat
   - ❑ fals, fals
3. Care dintre serviciile de mai jos NU este util intr-o soluție de webmail?
   - ❑ SMTP
   - ❑ IMAP
   - ❑ HTTP
   - ❑ SSH
4. Care dintre cele de mai jos NU reprezintă o tehnologie relevantă pentru serviciul web:
   - ❑ URL
   - ❑ HTTP
   - ❑ HTML
   - ❑ FTP
5. Care dintre următoarele este o aplicație server SMTP?
   - ❑ Postfix
   - ❑ Firefox
   - ❑ Outlook
   - ❑ Apache
6. Într-un client web (broswer) este tastată adresa:

```
ftp://bestman:none@test.com/lista/admisi.txt
```

   Care dintre următoarele afirmații este adevărată?
   - ❑ va fi inițiată o conexiune pe portul 80 către serverul test.com
   - ❑ va fi descărcat fișierul /test.com/lista/admisi.txt

- ❑ se va folosi pentru autentificare parola "bestman"
- ❑ pentru stabilirea conexiunii utilizatorul "bestman" să trebuie să existe pe test.com

7. Pentru conectarea la un server SSH este necesară cunoașterea adresei IP sau a numelui DNS a serverului. Filtrarea mesajelor spam se poate face numai pe serverul de mail, nu și pe client
   - ❑ adevărat, adevărat
   - ❑ adevărat, fals
   - ❑ fals, adevărat
   - ❑ fals, fals
8. Care din următoarele NU este un client web?
   - ❑ Firefox
   - ❑ Internet Explorer
   - ❑ wget
   - ❑ Thunderbird
9. Care dintre următoarele aplicații NU poate fi folosită pentru conectarea la un server web:
   - ❑ telnet
   - ❑ ssh
   - ❑ wget
   - ❑ curl
10. Comunicația între două servere de mail se realizează folosind:
    - ❑ POP3
    - ❑ HTTP
    - ❑ SMB
    - ❑ niciuna dintre variante

Capitolul 10

Elemente de securitate

There are two kinds of cryptography in this world: cryptography that will stop your kid sister from reading your files, and cryptography that will stop major governments from reading your files.

Bruce Schneier

Ce se învață din acest capitol?

- Problematica securității IT
- Elementele unei politici de securitate
- Securizarea sistemului
- Controlul accesului
- Securitatea sistemului de fișiere
- Forme de atac de rețea
- Criptare și firewall-uri
- Monitorizare

10.1 Problematica securității

Securitatea la nivelul calculatoarelor, în particular, și a domeniului IT, în general, a căpătat un interes deosebit în ultimele decenii. Deși un concept discutat din cele mai vechi timpuri, securitatea a căpătat noi forme, ramuri de proiectare și implementare odată cu dezvoltarea sistemelor de calcul și îndeosebi a Internetului. Pe o planetă în care peste 1.5 miliarde de oameni sunt conectați la Internet (circa 25% din populație)[1] securitatea și integritatea informației au o relevanță deosebită. În domeniul IT, domeniul securității a cunoscut o dezvoltare alertă, cu multe subdomenii și specialiști. Se discută despre ingineria securității, despre criptografie, gestiunea riscului, securitatea rețelei, inginerie socială, viruși și antiviruși, scrierea de cod sigur (secure coding).

[1] http://www.internetworldstats.com/stats.htm

În ciuda faptului că domeniul este unul vast, nivelul de cunoaștere și înțelegere a publicului larg în privința securității este destul de scăzut. De-a lungul ultimilor ani, multe teme tehnice specifice domeniului securității au părăsit domeniul IT, fiind preluate de ziare, jurnale TV, sau de industria cinematografică. Din păcate procesul nu a fost, cel mai adesea, unul ce încerca aducerea în sfera publică a conceptelor de securitate, ci mai ales unul ce speculează senzaționalul prin ignorarea constrângerilor lumii reale, ducând la promovarea unor noi mituri ale erei IT.

De multe ori se consideră normale "performanțele" hackerilor din filme, care reușesc să compromită securitatea unui sistem în câteva secunde. Exemple de "scenarii" care ajută la răspândirea acestui fenomen sunt scrierea de viruși pentru sisteme de operare extraterestre (vezi "Ziua Independenței", 1997) sau folosirea unui ecran 3D pentru virusarea unui sistem (vezi "Swordfish", 2001). Există, însă, și exemple pozitive în lumea cinematografică, precum folosirea **`nmap`** și SSH pentru compromiterea unei rețele electrice computerizate (vezi "Matrix", 1997[1]).

Capitolul de față nu își propune să ofere cititorului sfaturi despre obținerea unei puteri nemărginite în controlul tuturor sistemelor electronice, ci să aducă un nivel minim de ordine în domeniul populat de mituri al securității IT. A deveni cu adevărat un profesionist în securitate este o carieră, după cum spunea și Eric S. Raymond: *"Being able to break security doesn't make you a hacker anymore than being able to hotwire cars makes you an automotive engineer."*

10.1.1 Principii de bază

Securitatea calculatoare are drept scop protejarea informațiilor de la accese neautorizate, furt, corupere și alte riscuri. Domeniile securității se referă atât la mecanismele folosite pentru a obține accesul sau a corupe informațiile cât și la protejarea acestora. Forma generală de risc în securitatea IT se numește atac de securitate.

O primă clasificare a riscurilor de securitate distinge trei tipuri de atacuri:

- atacuri venite din Internet (cu o rată de succes redusă);
- atacuri inițiate din rețeaua locală;
- atacuri generate de pe aceeași mașină, acestea din urmă având un impact mult mai însemnat decât primele.

Deși cu gradul de risc cel mai ridicat, atacurile inițiate de utilizatorii serverului țintă sunt deseori tratate în grabă și unitar.

Domeniul securității dispune de un set de principii generale. Acestea nu sunt aplicabile doar în cadrul domeniului IT, ci în domeniul securității în general. Aceste principii nu sunt exhaustive, dar urmărirea acestora facilitează proiectarea și implementarea unui sistem cu nivel de siguranță cât mai ridicat.

Privit din perspectiva unui sistem IT, o soluție de securitate trebuie să includă atât o politică de securitate, ce definește drepturile și responsabilitățile utilizatorilor, cât și

[1] http://nmap.org/images/matrix/

specificații ale asigurării securității fizice, ale componentelor sistemului de operare, ale aplicațiilor locale, precum și ale serviciilor de rețea.

Unul dintre principiile fundamentale ale securității, fie ea IT sau de orice altă natură, este: **securitatea unui sistem este egală cu securitatea celei mai slabe verigi**.

Altfel spus, degeaba îți pui ușă ultra-performantă dacă nu folosești cheia, sau aplicat în domeniul IT: nu are rost să cheltui sume enorme de bani pe sisteme de securitate, dacă utilizatorii folosesc drept parole propriul nume, sau își țin parola lipită pe monitor.

În orice sistem există entități active (utilizatori, procese) și drepturi pe care acestea le au asupra sistemului. În condițiile asigurării securității sistemului, fundamental este **principiul celui mai mic privilegiu**. Acest principiu impune alocarea drepturilor minime entităților active: doar ceea ce este nevoie.

Un administrator de sistem va avea acces la întreg sistemul și resursele acestuia. Un utilizator obișnuit va avea acces doar la acele resurse care îi sunt necesare. Câteva de mecanisme care respectă acest principiu sunt:

- folosirea unui director de tip home cu drepturi complete pentru fiecare utilizator;
- folosirea comenzii **sudo** și fișierului `/etc/sudoers` pentru a conferi anumite privilegii utilizatorilor; (vezi secțiunea 10.3.3)
- separația între informațiile din fișierul `/etc/passwd` și `/etc/shadow`; (vezi secțiunea 10.2.3)
- folosirea **doar** a dreptului de execuție pe directoare când se dorește parcurgerea acestora; (vezi secțiunea 10.2.4)

Acest principiu este legat de **principiul limitării drepturilor** (vezi secțiunea 10.3.3). Întrucât resursele sistemului sunt limitate, un utilizator poate ajunge să folosească o foarte mare parte a acestora în dăuna altor utilizatori sau a sistemului de operare, putând conduce ușor la suspendarea funcționării acestuia. În general, limitarea drepturilor previne atacuri de tipul **DoS**[1] (*denial-of-service*).

Un utilizator poate crea foarte multe procese ducând la ocuparea memoriei (*fork bomb*[2]), un atac de rețea poate deschide multe conexiuni[3] care ocupă memorie, un utilizator poate ocupa spațiu pe disc și împiedica folosirea acestuia. Soluția este configurarea sistemului să limiteze drepturile utilizatorilor sau proceselor sistemului pentru a preveni suprautilizarea resurselor sistemului. Spre exemplu, folosirea cotelor[4] previne supraîncărcarea spațiului de pe disc.

Asocierea dintre o entitate activă (denumită și **subiect**) și drepturile pe care aceasta le deține asupra unor resurse (denumite și **obiecte**) se numește **controlul accesului** (*access control*). Sistemele de operare folosesc diferite mecanisme pentru a stabili când, cât de mult, cum și dacă poate un proces (subiect) să folosească o resursă (obiect).

O politică de securitate trebuie să stabilească un **compromis între gradul de flexibilitate a serviciilor IT și nivelul de securitate dorit**. Luate ad literam, cerințele de securitate ar presupune izolarea totală a sistemului de lumea exterioară, dar cum o

[1] http://en.wikipedia.org/wiki/Denial_of_service
[2] http://en.wikipedia.org/wiki/Fork_bomb
[3] http://en.wikipedia.org/wiki/SYN_flood
[4] http://en.wikipedia.org/wiki/Disk_quota

astfel de abordare duce la limitarea funcționalității, cel mai adesea securitatea unui sistem este definită ca un set de metode de protecție menite să descurajeze și să întârzie atacatorul.

Asigurarea flexibilității unui sistem presupune adăugarea de noi funcționalități. Aceasta se traduce, în general, în crearea unei aplicații complexe. Se spune "complexitatea este dușmanul securității"[1]. Un principiu de bază pentru asigurarea securității este **simplitatea**[2]. Orice nouă caracteristică adăugată unui sistem sau unui program poate însemna introducerea unei noi vulnerabilități. Se estimează că într-o companie specializată pentru dezvoltarea de programe, se introduc zeci de defecte (*bug-uri*) la fiecare 1000 de linii de cod. În dezvoltarea software, opusul simplității se numește *feature creep*[3]: adăugarea de funcționalități care nu sunt necesare.

Un sistem cu nivel foarte bun de securitate se poate obține mai ușor dacă planificarea acestuia ține cont de securitate decât dacă mecanismele de securitate sunt implementate după o perioadă în care acesta a fost dezvoltat. Planificarea unui sistem ținând cont de securitate (*design with security in mind*) este un factor decisiv pentru obținerea unui sistem cât mai sigur.

Dacă în proiectarea unei aplicații, a unei rețele sau a unui sistem, se ține cont de posibilile atacuri, de principiile de bază ale securității, de mediul și contextul de utilizare a aplicației, probabililitatea obținerii unui sistem vulnerabil este scăzută. De asemenea, orice noi mecanisme de asigurare a unui nivel suplimentar de securitate vor fi mai ușor adăugate sistemului. Ingineria sofware folosește denumirea de *secure by design*[4]. Un exemplu este sistemul de operare OpenBSD[5], care pune accent pe obținerea unui sistem cât mai sigur sigur.

Securitatea unui sistem nu este o finalitate, ci un proces. Un sistem nu va fi niciodată perfect sigur, cât timp acesta este folosit. Securitatea unui sistem presupune un șir de acțiuni continue pentru a preveni posibile atacuri. În general, securitatea se traduce în resursele pe care le poate investi un atacator față de resursele pe care le investește cel care protejează sistemul (bani, timp, personal).

În cartea "Secure Coding: Principles and Practices" [8] (capitolul 2 "Architecture"), se menționează că, într-un interviu de angajare, răspunsul potrivit la întrebarea "Cât de sigură poți să faci aplicația mea?" este "Cât de sigură vrei să fie?". Răspunsurile "Pot să o fac sigură în fața oricărui atac." sau "Pot să o fac cât de sigură se poate." ar trebui să rezulte în eliminarea candidatului din lista de potențiali angajați. Un sistem poate fi mai sigur decât un alt sistem, dar acest lucru se realizează cu **un număr crescut de resurse, stres și funcționalități obscure**. Întotdeauna trebuie cântărit cât de sigur se dorește a fi un sistem la câte resurse pot fi investite.

Nu se poate obține un sistem perfect sigur, dar un sistem trebuie menținut la un nivel de securitate. Operațiuni periodice de mentenanță, monitorizare, actualizare, verificare sunt necesare pentru asigurarea unui nivel dorit de securitate. O divizie responsabilă cu asigurarea securității unui sistem va trebui să considere în permanență **exteriorul ca fiind o zonă nesigură**. Tot ceea ce nu face parte din sistemul controlat trebuie

[1] http://www.schneier.com/news-038.html
[2] http://en.wikipedia.org/wiki/KISS_principle
[3] http://en.wikipedia.org/wiki/Feature_creep
[4] http://en.wikipedia.org/wiki/Secure_by_design
[5] http://www.openbsd.org/

considerat ca o zonă cu potențiale riscuri sau atacatori. Orice vulnerabilitate a sistemului trebuie reparată la timp pentru a nu permite unui potențial atacator să o folosească.

Depinzând de domeniu, pot exista și alte principii pentru asigurarea securității. Principiile prezentate mai sus reprezintă câteva din principiile de bază ale securității. Aceste principii nu sunt complete și nu sunt aplicate o singură dată. Pentru a obține un sistem cu nivel cât mai bun de securitate, principiile de bază trebuie revizuite, mecanismele folosite trebuie îmbunătățite și, în general, trebuie executate acțiuni periodice de evaluare a securității sistemului.

10.1.2 Termeni

Domeniul securității dispune de o serie de termeni care descriu la nivel general conceptele de bază folosite. Înțelegerea acestor termeni este fundamentală pentru orice specialist în IT și mai ales pentru cei care doresc să urmeze o carieră în domeniul securității.

Mai jos sunt prezentați o parte din termenii de bază ai domeniului:

- **amenințare** (*threat*) se referă la orice eveniment sau circumstanță cu potențial de a produce pagube unui sistem prin access neautorizat, distrugerea sau modificarea datelor sau denial of service; amenințările provin ca urmare a acțiunilor umane sau a cauze naturale;
- **autentificare** (*authentication*) înseamnă verificarea identității unui utilizator, proces sau dispozitiv, în general ca cerință pentru a permite acestuia accesul la resursele sistemului;
- **autorizare** (*authorization*) este oferirea sau respingerea drepturilor de acces pentru un utilizator sau proces;
- **confidențialitate** (*confidentiality*) este scopul securității care solicită protejarea datelor de la încercări intenționate sau accidentale de vizualizare/citire; confidențialitatea se referă atât la datele aflate pe un dispozitive de stocare, cât și la cele aflate în procesare sau în tranzit;
- **control acces** (*access control*) înseamnă permiterea folosirii autorizate a unei resurse, simultan cu prevenirea folosirii neautorizate sau într-un mod neautorizat;
- **integritate** (*integrity*) este scopul securității care solicită protejarea de la încercări intenționate sau accidentale de alterare a integrității datelor; **integritatea datelor** presupune ca acele date să nu fi fost alterate/modificate într-un mod neautorizat; la fel ca și confidențialitatea se referă atât la date aflate pe dispozitive de stocare, cât și la date aflate în procesare sau în tranzit;
- **denial of service** este un tip de atac care are ca efect prevenirea accesului autorizat la o resursă sau întârzierea operațiilor critice în raport cu timpul;
- **identitate** (*identity*) reprezintă o informație unică în cadrul unui domeniu de securitate, recunoscută ca o entitate unică în acel domeniu;
- **risc** (*risk*) reprezintă probabilitatea ca o anumită amenințare să exploateze o vulnerabilitate a sistemului și impactul care ar rezulta în urma acestei acțiuni;

- **vulnerabilitate** (*vulnerability*) se referă la o slăbiciune la nivelul proiectării, implementării sau operării sistemului, care poate fi declanșată intenționat sau accidental, rezultând într-o violare a politicii de securitate a sistemului.

Există o serie de confuzii de termeni de nume pentru persoanele implicate în domeniul securității. De multe ori noțiunea de *hacker* este folosită generic pentru a cataloga atacatorii de rețea deși, mai corect, denumirea corectă este cea de *cracker* (sau *black-hat hacker*). Următoarea listă reprezintă denumirile uzuale folosite și semnificațiile acestora:

- **hacker** este, în sensul său inițial, un programator caracterizat prin curiozitate și dorință de rezolvare a problemelor existente (de unde noțiunea de *hacking*[1]; în sensul securității, noțiunea de hacker poate avea două înțelesuri: *white-hat hacker* (*ethical hacker*) sau *black-hat hacker* (*cracker*);

- **cracker** (sau **black-hat hacker** este specializat în atacatarea unor sisteme pentru obținerea accesului; motivația poate fi financiară, politică, sau pentru distracție; scopul este, în general, distrugerea, furtul sau alterarea informației;

- **script kiddie** este un termen peiorativ folosit pentru a descrie persoanele fără cunoștințe tehnice deosebite pentru a fi considerați **hackeri** dar care folosesc scripturi sau programe scrise de alții pentru a ataca sisteme de calcul și rețele;

- **ethical hacker** (sau **white-hat hacker** este un expert în securitate, specializat în metodologii de testare a vulnerabilității unui sistem sau a unei rețele pentru a asigura un nivel cât mai bun de securitate; în general, ethical hackerii aplică metode similare crackerilor, dar cu scopul final de testare și securizare a sistemului atacat.

10.2 Securizarea sistemului

Securitatea sistemului se referă la mijloacele prin care se poate proteja un sistem de calcul. Atacurile pot veni din exterior (din Internet, sau din rețeaua locală), pot veni din interior (de la utilizatori) sau pot fi cauze naturale (căderea tensiunii poate conduce la pierderea unui hard disk sau la coruperea datelor).

Protejarea sistemului nu se referă doar la protecția sistemului de operare și a aplicațiilor ce rulează peste acesta ci și la protecția fizică a sistemului: poziționarea în zone sigure, folosirea de uși metalice, prevenirea accesului persoanelor neautorizate.

10.2.1 Securitatea sistemului de operare

Securitatea aplicațiilor trebuie să pornească de la asumarea gradului de risc dat de sistemul de operare. Dintre componentele sistemului de operare, o mare parte a atacurilor încearcă să exploateze limitări ale implementărilor de separare a drepturilor de acces la memorie sau la sistemul de fișiere.

[1] http://en.wikipedia.org/wiki/Hacker_(programmer_subculture)

Un sistem de operare este sigur dacă resursele acestuia (zone de memorie, dispozitive de intrare/ieșire, fișiere etc.) sunt accesate în mod valid de entitățile active (în general procese). Accesul valid este asigurat de nucleul sistemului de operare, ce acționează ca un intermediar între utilizatori și componentele hardware.

Pornind de la sistemele de operare simple (cele pentru PDA) până la sistemele complexe ce rulează pe calculatoarele personale, securitatea oferită de nucleu se bazează pe suport hardware. Procesoarele oferă cel puțin două niveluri de privilegii:

- un nivel de privilegii pentru operații obișnuite;
- un altul pentru accesul la instrucțiuni privilegiate: accesul la zone de memorie rezervată; accesul la dispozitivele hardware etc.

Accesul la aceste instrucțiuni privilegiate este permis doar nucleului.

Securitatea memoriei

În sistemele de operare moderne, securitatea memoriei este strâns legată de mecanismul de memorie virtuală. Fără a intra în detalii, este suficient de precizat faptul că sistemele de operare asigură fiecărui proces un spațiu dedicat de memorie virtuală (denumit și spațiu de adrese) care este mapat/asociat unor zone din memoria RAM. Zonele de memorie RAM peste care este mapat spațiul de adrese al unui proces nu poate fi accesat de către alte procese.

Problemele de securitate la nivelul memoriei țin de detectarea corectă a încercărilor de accesare a unor zone de memorie care nu se află în spațiul de adresă al procesului respectiv, dar și a încercărilor de schimbare a unor zone de memorie în spațiul propriu de adrese, pe parcursul rulării procesului.

Fiecare proces are propriul său spațiu de memorie virtuală mapat peste un spațiu de memorie fizică printr-o tabelă de translatare ce aparține procesului. Pentru un proces care încearcă să acceseze o adresă de memorie pentru care nu există o translatare către o adresă fizică (adică pentru care nu există pagina respectivă de memorie în RAM), procesorul va verifica la nivel hardware dacă respectiva adresă se află în spațiul de adrese al procesului. Dacă se află, înseamnă că respectiva pagină de memorie a fost evacuată pe disc (swap) și trebuie adusă în RAM. În caz contrar va fi generat un semnal de pagină (de memorie) invalidă (denumit page fault[1]), pe baza căruia nucleul va putea decide suspendarea sau terminarea procesului în cauză. În sistemele Unix procesul primește semnalul `SIGSEGV`, și are ca efect terminarea procesului cu mesajul *"Segmentation fault"*.

Toate aceste operațiuni cad în sarcina procesorului și sunt realizate la nivel hardware.

Protecția memoriei este o componentă esențială a sistemelor multitasking (sisteme ce oferă posibilitatea mai multor procese de a se afla în stare de execuție în același timp). Din punct de vedere hardware, primele procesoare pentru calculatoarele personale ce oferă suport pentru protecția memoriei sunt procesoarele din familia Intel 80386[2]. Folosind mecanisme împrumumate de la DOS, care rula pe sisteme fără suport

[1] http://en.wikipedia.org/wiki/Page_fault
[2] http://en.wikipedia.org/wiki/Intel_80386

hardware (8086, 80286), versiunile de Windows până la Windows 98 și Windows ME nu au oferit o protecție completă a memoriei.

10.2.2 Controlul accesului

Controlul accesului se referă la oferirea accesului autorizat și prevenirea acccesului neautorizat la sistem. Poate fi vorba de accesul fizic al unei persoane sau de accesul unui utilizator în cadrul sistemului de operare.

Securitatea fizică

Securitatea fizică include o gamă largă de parametri, de la controlul accesului personal până la asigurarea diferiților parametri optimi de funcționare.

O primă componentă a securității fizice presupune plasarea echipamentelor în locuri în care se poate asigura controlul accesului, pentru a preveni utilizarea neautorizată, distrugerea sau furtul.

Specificațiile de alimentare cu tensiune vor include evaluări ale consumului de putere pentru dimensionarea surselor neinteruptibile de putere (UPS-uri), sau a generatoarelor de tensiune.

Asigurarea temperaturii optime în centrele de operații ține de folosirea unor sisteme de aer condiționat sau de încălzire cu un grad ridicat de redundanță.

Există și cerințe de protecție împotriva incendiilor: o cameră cu servere va trebui să aibă, pe lângă sisteme de detecție și stingere a incendiilor, și podea și pereți din materiale rezistente la foc.

O bună politică de securizare va urmări plasarea echipamentelor în bunkere aflate la subsolul clădirilor, urmărindu-se astfel și protecția împotriva cutremurelor.

Securitatea la nivelul utilizatorilor

Una dintre componentele esențiale ale unei politici de securitate este securitatea la nivelul utilizatorilor. Securitatea utilizatorilor se bazează pe separarea resurselor fiecărui utilizator, precum și pe folosirea autentificării pentru protejarea accesului la resurse.

Principiul important de prevenire a atacurilor la nivelul utilizatorilor este principiul celui mai mic privilegiu. Orice operațiune trebuie executată dintr-un cont cu drepturi cât mai limitate.

Un administrator novice va folosi contul privilegiat inclusiv pentru efectuarea unor operațiuni uzuale. Exploatarea unor ambiguități ale variabilelor de mediu poate duce în acest caz la rularea accidentală a unei aplicații malițioase lăsată într-un spațiu public (gen `/tmp/`), rulare ce nu ar fi fost posibilă dintr-un cont de utilizator. Una dintre criticile aduse sistemelor din familia Windows a fost crearea unui cont implicit de tip Administrator care oferă drepturi depline utilizatorului sistemului, inclusiv instalarea

unor programe malițioase. Începând cu Windows Vista, acest lucru a fost combătut prin folosirea User Account Control[1].

10.2.3 Parole

Deși mecanismele de autentificare s-au diversificat, în continuare o pondere semnificativă o constituie perechile `<utilizator, parolă>`. O soluție alternativă, ce oferă un grad mult mai ridicat de scalabilitate, precum și de securitate, este folosirea de certificate. Conceptul ce stă la baza folosirii de certificate se numește PKI[2] (*Public Key Infrastructure*).

Numărul resurselor electronice fiind în continuă creștere, mulți utilizatori ajung să comită cel puțin una dintre erorile de căpătâi ale autentificării: alegerea de parole ușor de ghicit (spart), folosirea unei parole pentru mai multe conturi sau notarea parolelor în locuri nesigure.

Alegerea parolelor

Una dintre principalele probleme în alegerea parolelor este folosirea de parole prea simple. Mulți utilizatori folosesc numele propriu, numele soției, copiilor sau câinelui, o dată de naștere, numele echipei preferate. Aceste parole pot fi ușor ghicite printr-un atac de tipul dicționar.

Se spune că, o parolă eficientă:

- trebuie să aibă un număr de minim 7-8 caractere;
- trebuie să folosească atât minuscule, cât și majuscule și cifre;
- trebuie să includă cel puțin un caracter special (și nu doar pe ultima poziție în cadrul parolei);
- nu trebuie să fie un cuvânt din dicționar;
- nu trebuie să fie un nume de persoană.

Cu toate acestea, **parolele trebuie să fie ușor de reținut**. În această situație, cea mai la îndemână soluție este alegerea unei fraze pe baza căreia se poate deduce parola (vezi Ross Anderson – Security Engineering, secțiunea 3.3.3 [3]). *"Come to the dark side, we have cookies!"* poate fi folosită pentru a reține parola *"Cttds, whc!"*, după cum *"Social engineering bypasses all technologies, including firewalls"* poate genera *"S3bat,1f"*.

În cazul folosirii unei parole ce nu respectă recomandările de mai sus, atacurile bazate pe dicționar (încercarea parolelor ce aparțin unui set de cuvinte comune), sau cele bazate pe forță burtă (încercarea tuturor combinațiilor de caractere) pot avea rezultate aproape imediate. Există numeroase aplicații ce simulează astfel de atacuri, una dintre cele mai renumite fiind **John the Ripper**[3].

---

[1] http://en.wikipedia.org/wiki/User_Account_Control
[2] http://en.wikipedia.org/wiki/Public_key_infrastructure
[3] http://openwall.com/john/

De la conturi triviale de download până la conturile de email, sau la conturile bancare electronice, tot mai multe resurse solicită o autentificare pe bază de parolă. Este ideală **separarea conturilor prin parole individuale**. Cum acest lucru presupune memorarea unui număr ridicat de fraze, o soluție de compromis constă în folosirea unor parole comune pentru servicii similare cu miză de securitate redusă. Astfel, nu este prudentă folosirea PIN-ului cardului de bancă pentru contul de mail; un atac de forță brută ar dura aproximativ o microsecundă. Se poate însă folosi aceeași parolă pentru mai multe site-uri de descărcare de software.

Deși folosirea unor parole comune pentru servicii electronice cu miză mică reduce numărul parolelor, acestea pot fi însă, destul de multe. În plus, politicile de securitate solicită schimbarea periodică a parolelor: timpul de viață recomandat pentru o parolă este de șase luni. În măsura în care parolele folosite nu pot fi memorate, vor trebui notate. În mod evident, notarea parolelor nu trebuie făcută pe o foaie cu acces public. O soluție pentru parolele rar folosite constă în păstrarea lor centralizată. Există numeroase aplicații menite să protejeze printr-o singură parolă o bază de date de parole (spre exemplu **KeePass**[1].

Conștientizarea importanței unei parole nu ține doar de protejarea unor resurse personale, ci de gradul de securitate a întregului sistem, precum și a rețelei locale. Altfel spus, într-o rețea în care utilizatorii folosesc parole triviale, riscul nu este doar unul personal. Un atacator poate să compromită din Internet un cont local, transformând un atac de la distanță într-unul generat de pe aceeași mașină, ceea ce îi crește mult șansele de succes. Acesta este și motivul pentru care au evoluat și mecanismele prin care administratorii de rețea pot forța folosirea unor parole eficiente.

Atât rețelele Windows, cât și cele Linux oferă suport pentru securitatea bazată pe **jetoane hardware** (*security token*[2]) ce își regenerează parola la fiecare 30 sau 60 de secunde. Odată sincronizat ceasul serverului cu ceasul jetonului hardware, utilizatorul va folosi numele său de cont și parola afișată de jeton la momentul autentificării. Prețul unei astfel de soluții este de 5-10 euro per jeton hardware, fiecare jeton trebuind reconfigurat la 6 luni. Trebuie remarcat totuși că această soluție nu rezolvă riscurile de securitate fizică a parolei.

Gestiunea parolelor în Unix

Parolele erau, inițial, stocate în fișierul `/etc/passwd`, dar, pe măsură ce precalcularea și stocarea parolelor comune[3] a devenit posibilă, a apărut nevoia unui fișier separat, cu permisiuni mai stricte, în care sunt stocate parolele. Acesta se numește `/etc/shadow` și formatul său, cât și al `/etc/passwd`, este descris în secțiunea 3.3.1.

Utilitarul pentru schimbarea parolei în Unix este **`passwd`**. Utilizatorul privilegiat (root) poate schimba parola oricărui utilizator, inclusiv a sa. Un utilizator neprivilegiat poate schimba doar parola proprie. Înainte de a schimba parola îi este solicitată parola proprie pentru a preveni situația în care un utilizator uită o sesiune shell deschisă.

```
1 alina@valhalla:~$ passwd
2 Changing password for alina.
```

[1] http://keepass.info/

[2] http://en.wikipedia.org/wiki/Security_token

[3] Un astfel de atac se numește *dictionary attack*: http://en.wikipedia.org/wiki/Dictionary_attack

```
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Dacă se dorește obținerea unei parole ușor de reținut, dar care să fie relativ sigură, se poate folosi utilitarul **pwgen**. **pwgen** generează în mod implicit parole de 8 caractere (incluzând cifre):

```
alina@valhalla:~$ pwgen -N 1
uo3Pheis

alina@valhalla:~$ pwgen -N 2
Eeb7Aej9 Xoo0beir
```

Utilitarul permite "personalizarea" parolelor obținute. În exemplul de mai jos, parametrul `-n` precizează faptul că în parolă va fi inclusă cel puțin o cifră, parametru `-c` de va defini lungimea parolei dorite. Ultima cifră reprezintă numărul de parole ce trebuie generate.

```
alina@valhalla:~$ pwgen -n -c 7 3
ahngeZ5 aiv5Iec bie3poR
```

10.2.4 Securitatea sistemului de fișiere

Securitatea sistemului de fișiere urmărește asigurarea izolării utilizatorilor prin separarea fișierelor personale și prin definirea acțiunilor permise de utilizatori pe diverse secțiuni din ierarhia sistemului de fișiere.

Fiecare utilizator dispune de o intrare (un director) în sistemul de fișiere pentru care are drepturi depline. În Unix, acest director se găsește în `/home`. În Windows Vista, acest director este subdirector al `C:\Users`. Utilizatorul poate controla acțiunile permise pe fișierele conținute în respectivul director. Unele intrări în sistemul de fișiere nu pot fi folosite decât de utilizatorul privilegiat.

Controlul accesului

Controlul accesului la diverse elemente ale sistemului de fișiere diferă între sistemele Windows și cele Unix.

În Windows, sistemul de fișiere NTFS oferă de la versiunea 4.0 posibilitatea definirii unei liste de acces pentru fiecare utilizator din sistem. Această listă de acces este definită în raport cu șapte tipuri de acțiuni: *Full Control*, *Modify*, *Read & Execute*, *List*, *Read*, *Write*, *Special Permisions*. Subiectul drepturilor de acces pe un sistem de fișiere NTFS este tratat mai pe larg în studiul de caz al acestui capitol în secțiunea 10.6.1. Abordarea în cazul Windows este o formă a mecanismului de control a accesului[1].

Abordarea folosită în Unix pentru definirea drepturilor de acces la sistemul de fișiere este o matrice de drepturi.

Pentru a obține o matrice limitată, se folosesc trei trepte de privilegiu:

[1] http://en.wikipedia.org/wiki/Access_control_list

Tabelul 10.1: Matricea drepturilor de acces în Unix

	citire (r)	**scriere (w)**	**execuție (x)**
utilizator			
grup			
alții			

- **utilizatorul** (`user`) care deține fișierul;
- **grupul** (`group`) care deține fișierul;
- **alți utilizatori** (`others`).

Pentru fiecare dintre cele trei trepte de privilegiu sunt definite câte trei drepturi:

- **citire** (`read`) – permite vizualizarea conținutului unui director sau a unui fișier;
- **scriere** (`read`) – permite alterarea conținutului unui fișier sau director. Pentru un director, alterarea conținutului înseamnă posibilitatea ștergerii sau creării de fișiere;
- **execuție** (`execute`) – permite execuția unui fișier și permite parcurgerea unui director; dacă un director nu are drept de execuție, nu poate fi parcurs (nu se poate ajunge la fișierele și subdirectoarele conținute).

Având trei trepte de privilegiu și trei drepturi pentru fiecare treaptă, rezultă un total de 9 drepturi care pot fi configurate pentru un fișier dat. Forma liniarizată a matricei de acces (și a celor 9 drepturi) este afișată de comandă **`ls`** cu opțiunea de *long listing* (`-l`):

```
1 root@rosedu:~:~# ls -ld /var/svn/hfall/db/
2 drwxr-xr-x 6 hfall projects 4096 Jul 23 13:34 /var/svn/hfall/db/
```

În listing-ul de mai sus, utilizatorul asociat directorului `/var/svn/hfall/db/`, (`hfall`), are drepturi complete (`rwx`), iar grupul asociat (`projects`) și ceilalți utilizatori au drept de citire și execuție (`r-x`) (se poate vizualiza și parcurge directorul).

Forma liniarizată a matricei de acces pentru un fișier (afișată de comanda **`ls -l`**) este prezentată în figura 10.1.

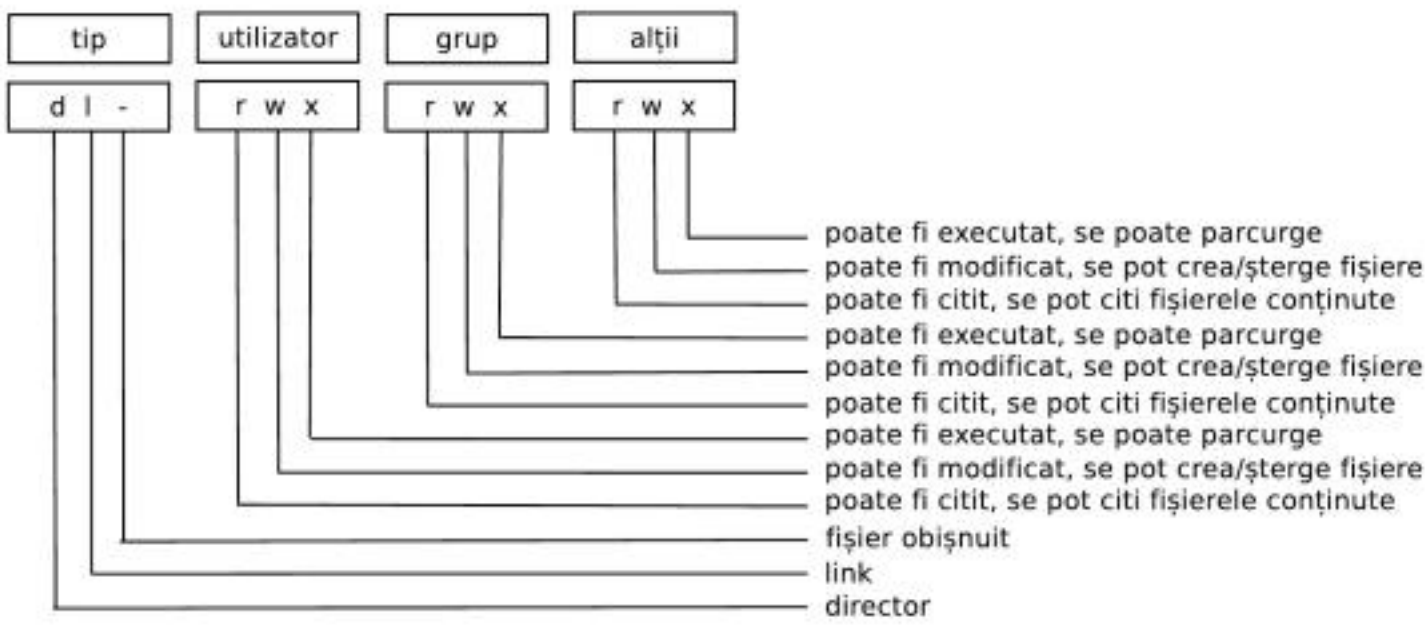


Figura 10.1: Permisiuni Unix

Pentru a specifica mai succint drepturile de acces la un fișier, utilizatorii avansați Unix preferă **forma octală a drepturilor de acces** la un fișier. În cadrul acestei forme, fiecare drept de acces este înlocuit cu un bit. Astfel, în cazul listing-ului de mai sus, drepturile de acces sunt:

```
1 rwx r-x r-x
2 111 101 101
```

Fiecare pereche de trei biți (asociată unei trepte de privilegiu) este tranformată într-o valoare octală. Forma finală, în octal, pentru drepturile de mai sus este, așadar `755`. În tabelul 10.2 sunt prezentate câteva exemple de asociere între forma literală și forma octală a dreputurilor de acces pe fișiere în Unix.

Tabelul 10.2: Permisiuni în format literal și octal

Format literal	Format binar	Format octal
r-xr-----	101 100 000	540
rw--w--wx	110 010 011	623
rwxr-xrw-	111 101 110	756
rwx--x--x	111 001 001	711
rw-r--r--	110 100 100	644

chown și chmod

În administrarea securității sistemului de fișiere în Unix, utilitarele de bază sunt **`chown`** și **`chmod`**. Primul oferă posibilitatea schimbării proprietarului și a grupului căruia îi aparține respectivul fișier, în vreme ce al doilea permite modificarea drepturilor.

```
1 root@kiwi:~# chown -R george:users test/
```

În exemplul de mai sus a fost efectuată o schimbare recursivă pentru toate fișierele și subdirectoarele directorului `test/`. Întreg conținutul directorului `test/` va aparține utilizatorului `george` și grupului `users`.

Dacă se dorește doar modificarea utilizatorului sau grupului, se folosește doar o parte a sintaxei:

```
1  razvan@valhalla:/tmp$ chown alina battleship
2  chown: changing ownership of 'battleship': Operation not permitted
3
4  razvan@valhalla:/tmp$ sudo bash
5
6  root@valhalla:/tmp# chown alina battleship
7
8  root@valhalla:/tmp# ls -l battleship
9  -rw-r--r-- 1 alina razvan 0 Sep 19 18:01 battleship
10
11 root@valhalla:/tmp# chown :shadow battleship
12
13 root@valhalla:/tmp# ls -l battleship
14 -rw-r--r-- 1 alina shadow 0 Sep 19 18:01 battleship
```

În exemplul de mai sus a fost schimbat utilizatorul și grupul fișierul `/tmp/battleship` în `alina`, respectiv `shadow`. Se observă că în Linux, un utilizator neprivilegiat nu

poate schimba deținătorul unui fișier. În Linux, comanda **chown** este folosită **doar** de către utilizatorul privilegiat. Pe alte Unixuri (spre exemplu, Solaris) comanda poate fi folosită și de un utilizator neprivilegiat.

Comanda **chmod** permite modificarea drepturilor de acces ale unui fișier. Comanda poate fi folosită doar de utilizatorul ce deține fișierul sau de utilizatorul privilegiat. Noile drepturi ale fișierului pot fi precizate în formă literală sau octală.

Fie fișierul `hello.c` de mai jos:

```
root@kiwi:~# ls -l
hello.c -rw-r--r--  1 razvan new 81 Oct  6 21:35 hello.c
```

Pentru schimbarea drepturilor se va folosi mai întâi forma literală. Drepturile pot fi precizate pentru oricare dintre cele trei niveluri de privilegiu: utilizator, grup, alți utilizatori printr-o singura literă: **u**, **g**, **o**. Drepturile pot fi:

- adăugate prin folosirea operatorului +;
- înlăturate prin folosirea operatorului −;
- precizate explicit prin folosirea operatorului =.

```
root@kiwi:~# chmod o+x hello.c

root@kiwi:~# ls -l hello.c
-rw-r--r-x  1 razvan new 81 Oct  6 21:35 hello.c

root@kiwi:~# chmod u=rx hello.c


root@kiwi:~# ls -l hello.c
-r-xr--r-x  1 razvan new 81 Oct  6 21:35 hello.c
```

În exemplul de mai sus a fost mai întâi adăugat pentru restul utilizatorilor dreptul de execuție, iar la al doilea pas s-a precizat explicit că drepturile utilizatorului trebuie să fie `rx` adică doar citire și execuție.

În format octal, pentru a acorda toate drepturile utilizatorului (`rwx`), drepturi de citire și execuție grupului (`r-x`), și doar drepturi de citire pentru restul utilizatorilor (`r--`) este suficientă o singură comandă:

```
root@kiwi:~# chmod 754 hello.c

root@kiwi:~# ls -l hello.c
-rwxr-xr--  1 razvan new 81 Oct  6 21:35 hello.c
```

umask

Pentru controlul drepturilor unei noi intrări în sistemul de fișiere se folosește un parametru de restricție numit **mască de creare** (*file mode creation mask*). Drepturile efective ce vor fi atribuite unei noi intrări în sistemul de fișiere se obțin prin efectuarea operației de ȘI logic între valoarea inversată a măștii și permisiunile implicite (`666` pentru fișiere și `777` pentru directoare). Reprezentată matematic, operația este `default_perm & ~mask`.

Comanda ce permite inspectarea și modificarea măștii se numește **umask**.

```
root@kiwi:~# umask
0022

root@kiwi:~# touch uso7_test1

root@kiwi:~# ls -l uso7_test1
-rw-r--r--  1 razvan razvan 0 Nov 10 17:28 uso7_test1

root@kiwi:~# umask 027

root@kiwi:~# touch uso7_test2

ro0t@kiwi:~# ls -l uso7_test2
-rw-r-----  1 razvan razvan 0 Nov 10 17:28 uso7_test2

root@kiwi:~# mkdir uso7_test_dir

root@kiwi:~# ls -ld uso7_test_dir/
drwxr-x---  2 razvan razvan 1024 Nov 10 17:29 uso7_test_dir
```

În listingul de mai sus, în primă fază masca are valoarea `0022`. Acest lucru înseamnă că un fișier nou creat (`uso7_test1` va avea drepturile

```
666 & ~022 = 666 & 755 = 110 110 110 & 111 101 101 =
        = 110 100 100 = 644 = rw- r-- r--
```

În ultimul pas, masca are valoarea `027`. Directorul `uso7_test_dir/` va fi creat cu drepturile

```
777 & ~027 = 777 & 750 = 111 111 111 & 111 101 000 =
        = 111 101 000 = 750 = rwx r-x ---
```

10.3 Întreținerea sistemului

După configurarea drepturilor de acces la nivelul sistemului de operare sau al sistemului de fișiere, unui sistem îi trebuie asigurată mentenanța. Aceasta presupune verificarea periodică a nivelului de securitate a sistemului, a jurnalelor, a acțiunilor utilizatorilor și proceselor. Prevenirea atacurilor este cea mai bună formă de asigurare a securității, iar un administrator de sistem profesionist va inspecta periodic sistemul și va lua măsurilor adecvate atunci când siguranța sistemului este periclitată.

10.3.1 Monitorizarea sistemului

Monitorizarea dinamică a sistemului

Monitorizarea dinamică (în timp real) a sistemului urmărește nivelul de încărcare a diferitelor resurse hardware. Astfel, pentru a determina gradul de încărcare a procesorului se poate folosi comanda **`uptime`**. Pentru a urmări nivelul de folosire a memoriei RAM se poate folosi comanda **`free`**.

Comanda **top** (vezi secțiunea 5.2.4) combină informațiile oferite de comenzile **uptime** și **free**.

```
1  top - 14:05:43 up  8:12,  2 users,  load average: 1.01, 1.02, 1.00
2  Tasks:  61 total,   3 running,  58 sleeping,   0 stopped,   0 zombie
3  Cpu(s): 99.7% us,  0.3% sy,  0.0% ni,  0.0% id,  0.0% wa,  0.0% hi,  0.0%
    si
4  Mem:    256740k total,   254200k used,     2540k free,    12680k buffers
5  Swap:   289128k total,     1444k used,   287684k free,    41356k cached
6
7    PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
8   2490 www-data  25   0 26204  18m  12m R 99.5  7.5 275:26.52 apache
9  14067 root      16   0  2064 1036 1852 R  0.3  0.4   0:00.02 top
10     1 root      16   0  1504  512 1352 S  0.0  0.2   0:00.73 init
11     2 root      34  19     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
12     3 root       5 -10     0    0    0 S  0.0  0.0   0:00.00 events/0
13     4 root       8 -10     0    0    0 S  0.0  0.0   0:00.00 khelper
14     5 root      14 -10     0    0    0 S  0.0  0.0   0:00.00 kacpid
15    31 root       5 -10     0    0    0 S  0.0  0.0   0:03.15 kblockd/0
16    41 root      20   0     0    0    0 S  0.0  0.0   0:00.00 pdflush
```

Prima linie conține (în același format) informațiile oferite de comanda **uptime**: ora curentă, intervalul de timp de la ultima inițializare a sistemului (în cazul exemplului 8 ore și 12 minute), numărul de utilizatori autentificați în sistem și încărcarea medie a sistemului pentru ultimele 1, 5 sau 15 minute. O încărcare supraunitară este echivalentă cu un sistem solicitat.

Liniile 4 și 5 oferă informațiile specifice comenzii **free**. Linia 4 descrie încărcarea memoriei RAM, iar linia 5 indică nivelul de utilizare a partiției de swap. Folosirea unei cantități însemnate de memorie din swap indică faptul că memoria RAM este insuficientă pentru cerințele sistemului.

În exemplul de mai sus se observă că procesorul este supraîncărcat, în vreme ce memoria RAM este bine dimensionată (pentru că memoria swap nu este folosită). Inspectând mai departe procesele cele mai relevante din punctul de vedere al consumului de resurse se observă că procesul apache a petrecut în procesor 275 de minute, adică mai mult de jumătate din timpul scurs de la ultima inițializare a sistemului. Se observă, de asemenea, că, în momentul rulării comenzii, procesul `apache` folosea 99.5% din procesor. Acesta este un comportament neobișnuit. Pentru un sistem bine configurat cele mai active procese vor reuși la nivelul unei luni de zile să ocupe 1-2 minute.

Concluzia analizei dinamice a sistemului de mai sus nu este că resursele hardware sunt insuficiente, ci că unul dintre servicii (mai exact serviciul de web) are un comportament anormal. Prima măsură ar trebui să fie oprirea procesului apache și inspectarea configurațiilor specifice acestui daemon.

Monitorizarea utilizării discului

O altă resursă importantă a unui sistem o reprezintă spațiul liber de pe hard-disk. Pentru aceasta trebuie monitorizat nivelul de ocupare a partițiilor definite pe disc, dar și dimensiunea unor directore și fișiere.

Spațiul liber pentru partițiile discului sistemului se poate vizualiza cu ajutorul comenzii **df**:

```
root@cursuri:/home/courses# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda1             4.6G  3.2G  1.2G  74% /
tmpfs                 126M  4.0K  126M   1% /dev/shm
/dev/hda6              32G   30G  1.2G  97% /home
```

În exemplul de mai sus spațiul pe disc este împărțit în două partiții: una folosită pentru rularea sistemului (montată ca `/`) și o a doua folosită pentru stocarea datelor utilizatorilor (montată ca `/home`). În acest fel, chiar dacă spațiul pe partiția `/home` este epuizat, acest lucru nu va afecta funcționarea sistemului.

În cazul în care se dorește inspectarea dimensiunii unui fișier anume se poate folosi comanda `ls` sau comanda `stat`:

```
root@rosedu:~# ls -lh /var/log/apache2/access.log
-rw-r----- 1 root adm 7.2M Sep 19 20:12 /var/log/apache2/access.log

root@rosedu:~# stat -c %s /var/log/apache2/access.log
7482956
```

Pentru a obține dimensiunea unui director se poate folosi comanda `du`:

```
root@cursuri:/home/courses# du -sh rc
155M    rc
```

Monitorizarea serviciilor de rețea

Pentru monitorizarea serviciilor de rețea disponibile pe stația locală este folosit utilitarul **netstat**.

Apelat fără niciun parametru, **netstat** va afișa o listă cu toate conexiunile active (porturile aflate în starea de listen se consideră inactive). O utilizare frecventă a acestui utilitar urmărește testarea funcționalității anumitor servere (HTTP, FTP etc.). În general, aceste servere așteaptă conexiuni TCP sau UDP pe anumite porturi.

Pentru a obține lista tuturor conexiunilor TCP, se poate folosi:

```
root@rosedu:~# netstat --tcp --all
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address   Foreign Address   State
tcp        0      0 *:ssh           *:*               LISTEN
tcp        0      0 *:smtp          *:*               LISTEN
tcp        0      0 *:git           *:*               LISTEN
[..]
```

Pentru a verifica funcționarea diferitelor servere instalate, care ascultă eventuale cereri de conectare, este utilă comanda:

```
root@rosedu:~# netstat --tcp --listening --numeric-ports
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address    Foreign Address   State
tcp        0      0 0.0.0.0:22      0.0.0.0:*         LISTEN
tcp        0      0 0.0.0.0:25      0.0.0.0:*         LISTEN
tcp        0      0 0.0.0.0:9418    0.0.0.0:*         LISTEN
```

```
7 tcp        0      0 127.0.0.1:3306   0.0.0.0:*         LISTEN
8 [..]
```

Se observă că sistemul mașina locală există servere care ascultă pe portul 22 (SSH), 25 (SMTP) sau 3306 (mysql). Așadar, server-ele SSH, SMTP și mysql sunt funcționale.

Pentru vizualizarea conexiunilor UDP, se poate utiliza parametrul `-u` sau `--udp`

Să presupunem că se descoperă un server care funcționează (se află în starea `LISTEN`) pe portul 22. Pentru oprirea acestui server putem opri procesul corespunzător. Înainte însă, trebuie aflat pid-ul acestui proces (opțiunea `-p` sau `--program`):

```
1 root@rosedu:~# netstat --tcp --listening --program --numeric-ports
2 Active Interne connections (only servers)
3 Proto Recv-Q Send-Q Local Address   Foreign Address  State    PID/Program
   name
4 tcp        0      0 0.0.0.0:80      0.0.0.0:*        LISTEN   5721/apache
5 [..]
6 tcp6       0      0 :::22           :::*             LISTEN   5515/sshd
7 tcp6       0      0 ::1:25          :::*             LISTEN   5505/master
```

Ulterior, se poate trimite un semnal `KILL` către procesul cu numărul `5515`.

```
1 root@rosedu:~# kill -KILL 5515
```

Monitorizarea activității utilizatorilor

Pentru a determina ce utilizatori sunt autentificati la un moment dat în sistem se foloseste comanda **`who`**:

```
1 root@cursuri:~# who
2 so       pts/0        Aug 10 13:50 (anaconda.cs.pub.ro)
3 ot       pts/1        Aug 10 14:05 (kiwi.cs.pub.ro)
```

Din exemplul de mai sus se poate deduce că în sistem sunt autentificați utilizatorii `so` și `ot`. Ambii utilizatori au acces în sistem prin emulatoare de terminal (`pts`), cu alte cuvinte prin intermediul unor sesiuni la distanță. Sesiunea utilizatorului `so` a fost inițiată la ora 13:50 de pe stația `anaconda.cs.pub.ro`, în vreme ce sesiunea utilizatorului `ot` a fost începută la 14:05 de pe sistemul `kiwi.cs.pub.ro`.

Pentru a determina ce utilizatori au folosit sistemul în ultima perioadă, se poate folosi comanda **`last`**:

```
1 root@swarm:~# last -5
2 root     pts/1        141.85.37.227    Sat Sep 19 20:24   still logged in
3 root     pts/1        188.27.105.225   Sat Sep 19 18:21 - 18:39  (00:17)
4 andreif  pts/2        81.181.250.24    Sat Sep 19 14:55 - 14:56  (00:00)
5 andreif  pts/2        81.181.250.24    Sat Sep 19 14:25 - 14:41  (00:15)
6 andreif  pts/2        81.181.250.24    Sat Sep 19 14:01 - 14:25  (00:24)
7
8 wtmp begins Tue Sep  1 08:01:55 2009
```

10.3.2 Jurnalizarea și gestiunea jurnalelor

Majoritatea serviciilor sistemului rulează ca daemoni (vezi secțiunea 5.3) și nu pot comunica direct cu utilizatorul. Utilizatorul interacționează cu serviciile prin intermediul

semnalelor și a fișierelor de configurare, iar serviciile oferă informații de stare, raportare și funcționare prin intermediul fișierelor jurnal (`server logs`).

Fișierele jurnal pot avea diverse formate, dar, în general, fiecare linie de tip jurnal precizează ora și data, adresa IP cu care serviciul a comunicat și o descriere a mesajului. Linia de mai jos este o parte dintr-un jurnal al unui server Apache în *Common Log Format*. În cazul de față este vorba de o conexiune inițiată de web crawler-ul de la Yahoo![1].

```
1 72.30.78.249 - - [19/Sep/2009:20:29:26 +0300] "GET /tag/picasa/ HTTP/1.0"
   200 6331 "-" "Mozilla/5.0 (compatible; Yahoo! Slurp/3.0; http://help.
  yahoo.com/help/us/ysearch/slurp)"
```

Pe un sistem Unix, jurnalele se găsesc, de obicei, în `/var/log/`:

```
1 root@swarm:~# ls /var/log
2 apache2           daemon.log.4.gz  kern.log        mail.log.4      syslog
3 apt               debug            kern.log.1      mail.log.5.gz   syslog
  .0
4 auth.log          dmesg            lastlog         mail.log.6      syslog
  .1
5 auth.log.0        dmesg.0          lpr.log         mail.log.6.gz   syslog
  .2.gz
6 [...]
```

Crearea jurnalelor

`sysklogd` este un utilitar care permite lucrul cu mesajele sistem și cele generate de kernel. Pe scurt, aplicațiile trimit diferite mesaje către **`sysklogd`** iar acesta, utilizând un fișier de configurare, depune aceste mesaje în anumite fișiere, le scrie în anumite console sau eventual le trimite către o altă mașină.

În Debian utilitarul **`sysklogd`** este disponibil în pachetul cu același nume.

Fișierul de configurare este `/etc/syslog.conf`. Acest fișier conține reguli care definesc modul în care sunt stocate mesajele de jurnalizare transmise de serviciile sistemului.

Deși este foarte util pentru monitorizarea sistemului, **`sysklogd`** prezintă și unele dezavantaje. De exemplu, o aplicație neglijentă ar putea genera foarte multe mesaje, ceea ce ar putea duce la umplerea discului și la instabilitatea întregului sistem. Trimirea mesajelor către o altă mașină, mai puțin importantă în rețea, ar rezolva această problemă.

Unele aplicații, precum serverul web Apache, nu folosesc **`sysklogd`**, ci implementează module proprii de jurnalizare.

Inspectarea jurnalelor

Pentru citirea jurnalelor se poate folosi orice utilitare de editare sau vizualizare, de la vi la less. Datorită faptului că mesajele recente sunt atașate la sfârșitul fișierului jurnal, un utilitar des folosit este **`tail`**.

[1] http://help.yahoo.com/l/us/yahoo/search/webcrawler/

Pentru inspectarea dinamică a jurnalelor se folosește comanda **tail** cu opțiunea `-f`. Comanda **tail** afișează în mod implicit ultimele 10 linii dintr-un fișier dat. Prin folosirea parametrului **-f** se asigură afișarea dinamică a liniilor adăugate la respectivul fișier.

Comanda de mai jos este un exemplu de folosire pentru inspectarea dinamică a activității serviciilor de e-mail:

```
root@cursuri:/var/log# tail -f /var/log/mail.log
```

Rotirea jurnalelor

Pentru a evita problema ocupării spațiului de pe disc de fișierele jurnal, sistemele Unix dispun de utilitarul **logrotate**. Acesta permite reutilizarea fișierelor de jurnalizare zilnic, săptămânal sau dacă au depășit o anumită dimensiune. În momentul în care condiția de timp sau spațiu a fost îndeplinită se efectuează o rotație.

O rotație înseamă salvarea conținutul jurnalului într-un fișier de backup cu un nume format din numele original la care se adaugă un număr. Fișierul de backup copia având un număr mai mare cu cât este mai veche (test.log.1, test.log.2 etc). La fiecare rulare, logrotate va șterge fișierul cel mai vechi și le va actualiza pe celelalte corespunzător. În general, fișierele de backup sunt arhivate și comprimate.

Mai jos sunt prezentate fișierele de jurnalizare pentru un server web Apache:

```
root@swarm:~# ls -l /var/log/apache2/
total 35092
-rw-r----- 1 root adm   8178409 Sep 19 20:45 access.log
-rw-r----- 1 root adm  10545752 Sep 13 03:35 access.log.1
-rw-r----- 1 root adm    549344 Jul 12 03:36 access.log.10.gz
-rw-r----- 1 root adm    712546 Jul  5 03:33 access.log.11.gz
-rw-r----- 1 root adm    845592 Jun 28 03:35 access.log.12.gz
[...]
```

logrotate este de obicei rulat zilnic de către serviciul de planificare periodică (`cron`). Fișierul său de configurare este `/etc/logrotate.conf`.

Alte utilitare de jurnalizare

Utilitarul **dmesg** poate fi folosit pentru afișarea informațiilor de la inițializarea sistemului. Acestea sunt mesaje ale nucleului și pot fi obținute și prin vizualizarea directă a jurnalelor de nucleu. Pentru configurațiile implicite, jurnalele de nucleu sunt păstrate în fișierul `/var/log/kern.log`.

10.3.3 Limitarea drepturilor

Sistemele Unix oferă suport în principal doar pentru două niveluri de privilegiu: utilizatori neprivilegiați și administrator (root). În practică este necesar un grad mai ridicat de flexibilitate în acordarea privilegiilor, pentru a permite utilizatorilor obișnuiți rularea unor aplicații privilegiate. În același timp, un utilizator obișnuit poate abuza de resursele care

îi sunt puse la dispoziție și poate, accidental sau intenționat, conduce la destabilizarea sau suspendarea sistemului.

De accea, un sistem de operare va oferi facilități pentru limitarea sau extinderea privilegiilor la nivel de aplicație sau la nivel de utilizator.

Folosirea atributelor setuid și setgid

O categorie aparte de aplicații o reprezintă utilitarele ce necesită resurse speciale din nucleu. Pentru a putea da posibilitatea și utilizatorilor neprivilegiați de a rula astfel de programe, identificatorul utilizatorului (uid) se schimbă pe parcursul execuției programului. La încheierea unei astfel de acțiuni se revine la identificatorul utilizatorului ce a lansat aplicația.

Într-un sistem Linux, acest lucru este realizat prin folosirea atributelor `setuid` și `setgid`. Aceste atribute sunt marcate ca permisiuni de acces pentru un fișier executabil. În momentul rulării acelui executabil, procesul creat rulează nu mai rulează cu drepturile utilizatorului care a lansat programul, ci cu drepturile utilizatorului ce deține fișierul; dacă acest utilizator este root, atunci procesul are drepturi privilegiate.

Pentru activarea atributelor `setuid` și `setgid` se folosește utilitarul **chmod**. Biții asociați celor două atribute se găsesc într-un număr în octal care precede cele 3 numere în octal asociate drepturilor de acces:

```
1  root@kiwi:~# ls -l a.out
2  -rwxr-xr-x  1 razvan razvan 13564 Jul  9 20:49 a.out
3
4  root@kiwi:~# chmod 2755 a.out
5
6  root@kiwi:~# ls -l a.out
7  -rwxr-sr-x  1 razvan razvan 13564 Jul  9 20:49 a.out
8
9  root@kiwi:~# chmod 4755 a.out
10
11 root@kiwi:~# ls -l a.out
12 -rwsr-xr-x  1 razvan razvan 13564 Jul  9 20:49 a.out
```

În exemplul de mai sus, prima rulare a comenzii **chmod** activează bitul asociat `setgid` (folosind valoarea octală 2). A doua rulare a comenzii **chmod** activează bitul asociat `setuid` (folosind valoarea octală 4). Alternativ, bitul asociat `setgid` poate fi activat folosind sintaxa `g+s`, iar bitul asociat `setuid` cu ajutorul sintaxei `u+s` ca argumente la **chmod**.

Trebuie avut în vedere că marcarea unui executabil cu dreptul de suid reprezintă un potențial risc de securitate. Dacă pe parcursul execuției utilizatorul reușește să forțeze terminarea anormală a programului înainte de refacerea uid-ului, se poate obține acces la un shell de root. Din acest motiv un pas important în securizarea unui sistem este determinarea tuturor executabilelor ce au activat atributul `setuid`, și dezactivarea acestuia de la aplicațiilor nefolosite. Aceast lucru se poate realiza cu ajutorul comenzii **find** (vezi secțiunea 12.6.3).

```
1  root@kiwi:~# find / -perm -4000 -o -perm -2000 -type f -print
```

Printre utilitarele care au nevoie de activarea atributului `setuid` pentru a putea fi rulate de către utilizatori obișnuiți se află **`ping`** și **`traceroute`**.

Schimbarea rădăcinii sistemului de fișiere (chroot)

Unul dintre mecanismele extrem de eficiente în securizarea unor servicii se numește **chroot**. Acesta presupune dimininuarea vizibilității unui serviciu în raport cu sistemul de fișiere.

Folosirea mecanismului **chroot** are rolul de reducere a riscurilor de securitate. Dacă un atacator obține acces cu ajutorul unei vulnerabilități într-un proces ce folosește **chroot**, acesta va avea acces limitat la sistemul de fișiere. Noul director rădăcină vizibil procesului poartă numele de *chroot jail*.

Pe de altă parte, folosirea acestui mecanism este destul de dificilă, datorită faptului că în general programele sunt compilate pentru a folosi biblioteci (vezi secțiunea 11.4). Din această cauză, bibliotecile trebuie incluse în directorul văzut de proces ca rădăcină. O altă soluție este compilarea statică a programelor, fără biblioteci.

Fie serverul de DNS **bind9**. Dacă se modifică scriptul de inițializare adăugând opțiunea `-t /var/dns`, serviciul va porni considerând `/var/dns/` ca rădăcină a întregului sistem de fișiere. Primul fișier căutat va fi `/etc/bind/named.conf`, astfel că înainte de lansarea serviciului va trebui copiat fișierul `/etc/bind/named.conf` în `/var/dns/etc/bind/named.conf`. Pe lângă acest fișier vor trebui copiate fișierele de zonă (necesare configurării serviciului de DNS), fișierele de jurnalizare, dar și unele fișiere speciale gen `/dev/null` sau `/dev/log`.

Majoritatea programelor ce implementează serviciile UNIX oferă posibilitatea rulării procesului cu sistemul de fișiere limitat. Dacă programul nu oferă această posibilitate, sau dacă se dorește testarea mecanismului, se poate folosi utilitarul **`chroot`**.

Utilitarul **`chroot`** este implementarea la nivelul shellului a mecanismului de schimbare a rădăcinii sistemului de fișiere văzut de un proces.

În exemplul de mai jos se rulează utilitarul chroot într-un nou director `/root/test`. Dacă nu se specifică ce comandă se rulează, implicit se execută interpretorul de comenzi precizat de variabila de mediu `SHELL`. Cum `/bin/sh` este o legătură către `/bin/bash`, este necesară copierea sa în viitorul sistem de fișiere.

```
1  root@kiwi:~# pwd
2  /root
3
4  root@kiwi:~# mkdir test
5
6  root@kiwi:~# mkdir test/bin
7
8  root@kiwi:~# cp /bin/bash test/bin/
9
10 root@kiwi:~# chroot /root/test
11 chroot: cannot run command '/bin/bash': No such file or directory
12
13 root@kiwi:~# ldd /bin/bash
14         libncurses.so.5 => /lib/libncurses.so.5 (0x4001c000)
15         libdl.so.2 => /lib/tls/libdl.so.2 (0x4005b000)
```

```
16          libc.so.6 => /lib/tls/libc.so.6 (0x4005f000)
17          /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
18  root@kiwi:~# mkdir test/lib
19
20  root@kiwi:~# cp /lib/libncurses.so.5 /lib/libdl.so.2 test/lib
21
22  root@kiwi:~# cp /lib/libc.so.6 test/lib
23
24  root@kiwi:~# chroot /root/test
25
26  root@kiwi:/# pwd
27  /
```

Se observă că, în primă fază, rularea comenzii **bash** prin **chroot** nu poate fi realizată. Motivul este absența bibliotecilor. Se afișează apoi lista bibliotecilor necesare **bash**, iar respectivele biblioteci sunt apoi copiate în viitorul sistem de fișiere. Comanda **chroot** poate fi acum executată, rezultând un prompt într-un sistem de fișiere ce are doar 2 directoare și 4 fișiere. Se pot totuși rula comenzile interne shellului **bash**.

sudo

Utilitarul **sudo** permite utilizatorilor neprivilegiați să ruleze unele comenzi având identitatea (uid-ul) altui utilizator (în general a utilizatorului root). Utilitarul **sudo** poate fi folosit pentru a oferi privilegii limitate anumitor utilizatori și a evita astfel folosirea contului root; privilegiile acestuia pot deveni o problema în cazul în care contul este compromis.

Controlul utilizatorilor care au dreptul să ruleze comenzi privilegiate, precum și lista exactă a comenzilor este păstrată în fișierul **/etc/sudoers**. Fișierul are în mod implicit doar drepturi de citire.

Pagina de manual menționează *The sudoers file should always be edited by the visudo*. Utilitarul **visudo** evită oferirea unor drepturi provizorii de scriere și minimizează astfel alte riscuri de securitate.

La lansarea unei comenzi **sudo**, după verificarea fișierului **/etc/sudoers** va trebui introdusă parola utilizatorului ce a lansat comanda **sudo**. Dacă se dorește rularea comenzilor privilegiate configurate **fără** solicitarea parolei (risc de securitate), se folosește tag-ul `NOPASSWD` în fișierul de configurare:

```
1  razvan  ALL=(ALL) NOPASSWD: ALL
```

Listingul de mai sus este o linie din fișierul `/etc/sudoers`. Utilizatorul `razvan` poate rula în numele **tuturor** utilizatorilor din sistem (`ALL=(ALL)`) orice comandă (`ALL`), fară a îi fi solicitată parola (`NOPASSWD`).

Pentru exemplul următor să presupunem că a fost dezactivat atributul `setuid` din cadrul executabilului `/bin/ping`. Pentru a permite utilizatorului `rookie` să folosească comanda **ping**, administratorul va trebui să adauge în fișierul `/etc/sudoers` următoarea linie:

```
1  rookie ALL=(ALL) /bin/ping
```

Dacă utilizatorul `rookie` încearcă rularea unui interpretor de comenzi cu drepturi privilegiate, operația eșuează:

```
rookie@kiwi:~$ sudo /bin/bash
Sorry, user rookie is not allowed to execute '/bin/bash' as root on
localhost.
```

Dacă se încearcă rularea comenzii **ping**, fară a folosi **sudo**, se obține din nou mesaj de eroare:

```
rookie@kiwi:~$ /bin/ping 141.85.99.1
-su: /bin/ping: Permission denied
```

Folosirea comenzii **sudo** impune parcurgerea fișierului `/etc/sudoers` și permite rularea executabilului `/bin/ping`:

```
rookie@kiwi:~$ sudo /bin/ping 141.85.99.1
PING 141.85.37.1 (141.85.37.1): 56 data bytes
64 bytes from 141.85.37.1: icmp_seq=0 ttl=64 time=0.5 ms
```

Rularea unei comenzi **sudo** fără a avea o corespondență în fișierul `/etc/sudoers` va fi jurnalizată ca un incident de securitate. Pentru a verifica lista comenzilor ce pot fi rulate ca `root` se poate folosi opțiunea `-l` a comenzii **sudo**:

```
rookie@kiwi:~$ sudo -l
User rookie may run the following commands on this host:
    (ALL) /bin/ping
```

Limitarea resurselor folosite de un proces sau utilizator

Utilitarul **ulimit** permite controlul resurselor alocate pentru un proces pornit din interpretorul de comenzi. Pentru inspectarea limitărilor curente se poate se folosește argumentul `-a`:

```
root@kiwi:~# ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 16382
max locked memory       (kbytes, -l) 64
[...]

root@kiwi:~# ulimit -n 2048

root@kiwi:~# ulimit -n
2048
```

În exemplu de mai sus a fost realizată o listare a tuturor limitărilor curente, pentru ca apoi să fie redefinit numărul maxim de fișiere deschise la 2048.

Dacă, în scopul prevenirii unui atac DoS local (cauzat de crearea unui număr de procese extrem de mare), se dorește limitarea numărului de procese pentru utilizatorul privilegiat, se recomandă adăugarea în fișierul `/root/.bashrc` a unei limite de procese:

```
ulimit -u 1024
```

Utilitarul **ulimit** permite limitarea drepturilor la nivelul unui proces fără a fi însă persistente. La repornirea sistemului, configurările realizate se vor pierde. Pentru a

asigura persistența schimbărilor, pe un sistem Linux se poate folosi fișierul `/etc/security/limits.conf`. Acest fișier reprezintă un mecanism eficient pentru limitarea resurselor disponibile unui utilizator sau unui grup de utilizatori.

Se pot defini două tipuri de limite, denumite limite software și limite hardware. La depășirea unei limite software se generează un mesaj de atenționare. La depășirea unei limitări hardware procesul în cauză este terminat.

Sintaxa fișierului se bazează pe definirea de reguli. O regulă are patru elemente:

- **ținta**: utilizatorii sau grupurile pentru care se aplică respectiva regulă;
- **tipul de regulă**: regulile pot fi definite ca software sau hardware;
- **tipul de limitare**: este precizat parametrul care este limitat;
- **limitarea**: valoarea la care se realizează limitarea.

Ca exemplu, se presupune că se dorește limitarea numărului de procese ale utilizatorului `rookie` la 35, iar la depășirea limitei de 20 de procese să fie generat un mesaj de avertisment. În plus se dorește limitarea numărului de autentificări **simultane** permise utilizatorului `rookie` la 10. Pentru aceasta vom adăuga în fișierul `/etc/security/limits.conf` următoarele trei linii:

```
1 rookie  soft    nproc           20
2 rookie  hard    nproc           35
3 rookie  -       maxlogins       10
```

Limitarea spațiului ocupat pe disc

Limitarea spațiului ocupat de fișierele unui utilizator se realizează prin folosirea de cote. În Linux, acest lucru este realizat cu ajutorul utilitarului **`quota`**. Întrucât configurarea acestui utilitar depășește domeniul acoperit de această carte, recomandăm cititorului să parcurgă unul din multiplele tutoriale din Internet legat de configurarea cotelor în Linux.

10.4 Atacuri de rețea

Privite ca forma principală de acțiune a crackerilor, atacurile de rețea nu trebuie să constituie cunoștințe interzise sau privite cu suspiciune. Înțelegerea modului în care un atac este produs, a condițiilor care facilitează producerea acestuia și a efectelor ulterioare sunt esențiale pentru a proteja o rețea de calculatoare sau pentru a defini politicile de securitate asociate. Cunoștințele și deprinderile legate de atacurile de rețea sunt necesare *ethical hacker*-ilor, persoane specializate în forme de testare a sistemelor informatice pentru securizarea acestora.

10.4.1 Tipuri de atacuri în rețea

Atacurile de rețea sunt clasificate în funcție de gravitate în trei categorii: atacuri de recunoaștere, atacuri DoS, și atacuri de obținere a accesului.

Atacurile de recunoaștere nu sunt atacuri propriu-zise, ci de obicei constituie faza inițială, de investigare, a unui atac (vezi secțiunea 10.4). Cel mai simplu atac de recunoaștere poate fi generat printr-un script ce inventariază stațiile dintr-o rețea folosind pachete de ping. Pentru a obține informații în legătură cu serviciile rulate pe o mașină se poate folosi o aplicație de scanat porturi (vezi secțiunea 10.4.3). Aplicațiile mai complexe de recunoaștere oferă nu numai lista serviciilor, dar și o listă a posibilelor vulnerabilități. Un astfel de utilitar este Nessus[1].

Atacurile DoS (*Denial of Service*) au ca scop congestionarea unei rețele, a unui server sau doar a unui serviciu, în scopul reducerii sau opririi funcționalității. Se poate distinge între atacurile clasice DoS, bazate în general pe *flooding*, și atacurile DoS distribuite, numite **DDoS**, ce au la bază un mecanism de propagare (un virus) pe un număr cât mai mare de destinații și sincronizarea atacului DoS către o țintă.

Un **flood** reprezintă un număr foate mare de pachete venite într-un interval scurt de timp. Un trafic ce poate fi interpretat drept flood la nivelul ruterului de acces în rețeua locală poate fi tratat drept trafic normal în nucleul Internetului. Pentru tehnologiile actuale un flood este definit astfel:

- la nivelul unui switch de nivel 2 un număr de 100.000-150.000 de pachete pe secundă;
- la nivelul unui ruter de acces (ce conectează o rețea locală) un flood are 8.000-12.000 de pachete pe secundă;
- în nucleul Internetului un trafic de milioane de pachete pe secundă poate fi tratat drept trafic legitim.

În ciuda capacității de prelucrare a unui volum de pachete de ordinul zecilor de mii pe secundă, complexitatea adăugată de modele de flood specifice poate reduce limita de pachete pe secundă la câteva sute.

ICMP flooding se bazează pe trimiterea unui număr foarte mare de pachete ICMP (de tip ping), consumând întreaga lățime de bandă disponibilă. Cel mai adesea atacul este prevenit prin filtrarea întreg traficului ICMP, cu costul pierderii funcționalități utilitarelor **`ping`** și **`traceroute`**.

Atacul **UDP flooding** are efecte puternice când folosește drept țintă porturile 7 și 19, adică serviciile de `echo` și `chargen`. Aceste servicii fiind rar folosite, în rețele locale pot fi oprite de firewall-ul de intrare în LAN, fără un impact real asupra funcționării rețelei.

Atacurile bazate pe flooding sunt în general combinate cu un atac de tip **spoofing**, prin care se generează adrese sursă fictive și diferite pentru pachetele din flood. În lipsa unui atac spoofing, o filtrare pe baza depășirii unui număr limită de pachete per sursă ar elimina atacul flooding. Atacul spoofing asigură în același timp ascunderea identității atacatorului. În plus, dacă destinația atacului bazat pe flooding nu este o sursă țintă ci adresa de difuzare a unei rețele, toate echipamentele din respectiva rețea vor încerca să răspundă unei surse ce nu există în realitate, atacul devenind un atac de tip DDoS și se va numi un **atac smurf**.

Atacurile DDoS pornesc de la puterea enormă de calcul disponibilă în rețelele actuale locale. Unele dintre aceste atacuri pot fi neintenționate, precum în cazul Slashdot effect. Numele vine de la vestitul site Slashdot, care a postat un link către un site cu capabilități

[1] http://www.nessus.org/nessus/

mai mici. Când foarte mulți utilizatori au încercat să acceseze respectivul site, efectul a fost echivalent cu un atac DoS.

Atacurile de obținere a accesului au impactul cel mai mare, deoarece prin obținerea accesului la un serviciu cel mai adesea pot fi compromise și alte servicii sau mașini.

O pondere importantă în atacurile de obținere a accesului o au atacurile bazate pe exploatarea vulnerabilității aplicațiilor web. Subiectul este unul amplu, iar cei ce vor să urmărească numeroasele metode de exploatare a riscurilor de securitate ar trebui să parcurgă cartea lui Joel Scambray, *Hacking Exposed Web Applications* [24].

Atacurile de obținere a accesului pot avea drept țintă și tehnologiile din spatele aplicațiilor web, un număr important de atacuri fiind direcționat împotriva bazelor de date. Cel mai cunoscut astfel de atac este **inserarea de cod SQL** (*SQL Injection*). Acest atac se bazează pe modul direct de interogare a bazei de date.

```
1 SELECT X from TABLE where user = $user_input AND pass = $pass_input
```

Presupunând scriptul de mai sus, dacă la introducerea în câmpul utilizator a unui nume valid de utilizator urmat `OR --` (în SQL `--` definește un comentariu), operația de selecție devine:

```
1 SELECT X from TABLE where user = $xxx OR -- AND pass = 'nu_conteaza'
```

și va fi validată pertru orice nume de utilizator aflat în baza de date. Mergând mai departe se poate introduce în câmpul utilizator `xxx OR 1=1 OR --`, aceasta garantând accesul indiferent dacă utilizatorul `xxx` există sau nu în baza de date.

Pentru a preveni un astfel de atac, interogarea bazei de date trebuie făcută prin funcții de bibliotecă și nu direct prin select.

Există aplicații ce oferă posibilitatea lansării unui număr mare de atacuri. O astfel de aplicație este metasploit[1]. După o fază de determinare a vulnerabilităților (efectuată, spre exemplu cu Nessus) se pot selecta vulnerabilitățile găsite și pot fi lansate simultan mai multe atacuri de obținere a accesului.

Dincolo de noțiunile tehnice legate de atacurile de rețea, nu trebuie pierdut din vedere riscul de securitate adus de utilizatorii neglijenți din rețea. Chiar și în cazul unor utilizatori responsabili, nu trebuie ignorate riscurile unui atac bazat pe inginerie socială. Nu este o întâmplare că cel mai cunoscut cracker este Kevin Mitnik, ale cărui atacuri s-au bazat în mare măsură pe o bună înțelegere a modului de gândire a utilizatorilor și a administratorilor din rețelele actuale.

10.4.2 Viruși, viermi, troieni

Un virus este un program sau doar o secvență de cod ce se atașează altor fișiere executabile fără cunoștința sau acceptul utilizatorului. Un virus include alături de mecanismele de execuție și modalități de replicare prin inserare în codul altei aplicații.

[1] http://www.metasploit.com

Clasificarea virușilor în funcție de modul de replicare distinge între mai multe tipuri de viruși, dintre care cei mai întâlniți sunt: viruși de email, de macro, bombe logice, viruși de boot, viermi și troieni.

Simpla prezență a unui fișier infectat pe un calculator nu este echivalentă cu infectarea sistemului. Pentru a infecta sistemul, un virus trebuie să fie executat cel puțin o dată. Unele aplicații rulează cod executabil în mod automat (fără informarea utilizatorului). Un exemplu este rularea fișierului `autorun.inf`[1] la montarea unui CD-ROM sau USB stick.

Email-ul este unul dintre modurile principale de propagare a virușilor. **Virușii de email** sunt conținuți în atașamentul emailului, deseori având extensia schimbată (cel mai adesea în extensie specifică imaginilor). Aplicația țintă a unui astfel de virus este clientul de email. Datorită numărului mare de utilizatori ai Microsoft Outlook, o mare parte a acestor viruși sunt dezvoltați special pentru această aplicație.

Odată executat un atașament de email infectat, virusul se încarcă în memoria RAM și va urmări să se multiplice. Pentru aceasta va căuta lista de adrese construite de clientul de email (*address book*). Majoritatea clienților de email creează lista de adrese în mod dinamic, fără consultarea utilizatorului, și nu o protejează prin criptare. Odată deschisă lista de adrese, virusul va folosi direct API-ul de trimis emailuri pentru a se multiplica. Exemplele de viruși de email sunt `MyDoom`, `I love you`.

O a doua categorie importantă de viruși o reprezintă **virușii de macro**. Un macro este o bucată de cod executabil, atașată unui fișier document. Aplicațiile țintă a acestor viruși sunt cele de gestionare a documentelor, gen editoare de text, foi de lucru, sau editoare de prezentări. Aceste aplicații în general vor atenționa utilizatorul înainte de deschiderea unui document ce conține macro-uri. Pentru a reduce gradul de risc, dacă la deschiderea unui fișier sunteți avertizați de existența unor macro-uri, închideți documentul și rulați un program antivirus pe respectivul fișier.

Virușii de boot nu reprezintă o amenințare majoră pentru sistemele actuale datorită dificultății de lansare în execuție. Pentru a lansa un astfel de virus, sistemul trebuie să încerce inițializarea de pe o partiție infectată. Impactul acestor viruși s-a diminuat odată cu dispariția floppy discurilor. În prezent virușii de boot se pot propaga prin CD-uri sau flash carduri.

Troienii (trojan horses) reprezintă aplicații (aparent legitime) ce conțin (sau instalează) un program malițios. Pentru a evita astfel de atacuri se recomandă instalarea aplicațiilor doar din surse sigure, precum și verificarea integrității fișierelor înainte de instalare.

Viermii (worms) sunt programe care folosesc vulnerabilități în diferite servicii din Internet pentru a se multiplica. Virușii de mail reprezintă o categorie de viermi.

Pentru a reduce impactul virușilor sunt importante atât măsurile luate la nivelul rețelei, cât și exersarea unor deprinderi din partea utilizatorilor. Este important ca accesul în rețea să fie protejat de un firewall (vezi secțiunea 10.5.1), ca serverul de email să aibă instalat un antivirus, să existe o politică de monitorizare a serviciilor etc. La nivelul utilizatorului este important să nu lanseze în execuție fișiere primite prin email, să nu deschidă atașamente primite de la necunoscuți, să mențină un antivirus actualizat pe stația de lucru etc.

[1] http://en.wikipedia.org/wiki/AutoRun#Attack_vectors

10.4.3 Scanarea porturilor

Un program de scanat porturi este o aplicație care inițiază conexiuni către toate porturile unei sistem țintă. Scopul este cel de a detremina care porturi sunt deschise, aflând astfel ce servicii sunt disponibile.

Programele de scanat porturi fac parte din setul de utilitare de bază folosite atât de crackeri (*black-hat hackers*) și ethical hackeri (*white-hat hackers*) pentru a detecta serviciile deschise pe un sistem și versiunea acestora. Dacă în urma scanării de porturi se detectează servicii potențial vulnerabile, se poate trece la folosirea unui exploit.

Sfârșitul anilor '90 în România a dus la o creștere dramatică a popularității programelor de scanat porturile, atât în rândul crackerilor cât și al administratorilor de rețea. Principala cauză a acestei schimbări a fost dezvoltarea rețelei metropolitane, precum și a diferenței foarte mari între costul traficului metropolitan și cel al traficului Internet. Astfel într-un interval relativ scurt numărul atacurilor asupra serverelor aflate în rețele publice (mai ales asupra celor aflate în universități) a crescut foarte mult.

Deși nu este ilegală, folosirea unui scaner de porturi poate fi interpretată drept un atac, și prin urmare este posibil să vă fie filtrat accesul din rețeaua din care au fost inițiate scanările. O astfel de restricționare este totuși destul de improbabilă.

Utilitarul nmap

Există numeroase aplicații de scanat porturile, una dintre cele mai cunoscute fiind **nmap**[1]. Datorită popularității de care se bucură în rândul administratorilor de sisteme Linux, aplicația a fost portată și pe platforme Windows[2].

nmap implementează mai multe metode de scanare, putând oferi informații despre porturile deschise, dar și despre porturile explicit filtrate. În plus față de informațiile referitoare la sericiile rulate, nmap pune la dispoziția administratorului de rețea metode de detectare a stațiilor active dintr-o rețea, de aflare a versiunii sistemului de operare, versiunile serviciilor active, utilizatorii ce rulează aceste servicii, precum și multe alte informații relevante pentru sistemul scanat.

Dintre cele opt tipuri importante de scanare, șase se referă la scanarea porturilor TCP, unul la scanarea porturilor UDP și unul la monitorizarea conectivității, această ultimă metodă încercând să determine doar dacă stația destinație este activă și dacă se poate ajunge la ea.

Opțiunea `-sP` (*Ping Scan*) permite detectarea stațiilor active într-o rețea locală:

```
1 root@cursuri:~# nmap -sP 141.85.37.0/24
2
3 Starting Nmap 4.62 ( http://nmap.org ) at 2009-09-23 16:21 EEST
4 Host csr.cs.pub.ro (141.85.37.1) appears to be up.
5 MAC Address: 00:09:6B:89:06:67 (IBM)
6 Host ns.catc.ro (141.85.37.2) appears to be up.
7 MAC Address: 00:17:31:49:3A:E4 (Asustek Computer)
```

[1] http://www.insecure.org/nmap

[2] http://www.sourceforge.net/projects/nmapwin

```
Host prof.cs.pub.ro (141.85.37.3) appears to be up.
MAC Address: 00:09:6B:89:05:24 (IBM)
Host info.cs.pub.ro (141.85.37.6) appears to be up.
MAC Address: 00:13:8F:78:42:2C (Asiarock Incorporation)
Host 141.85.37.7 appears to be up.
MAC Address: 00:50:56:9A:33:46 (VMWare)
Host 141.85.37.8 appears to be up.
MAC Address: 00:09:6B:89:06:67 (IBM)
```

Metodele de scanare a porturilor TCP sunt diferențiate prin prezența unui anumit flag într-un pachet TCP. Opțiunea `-sS` (*TCP SYN scan*) este opțiunea implicită care folosește flagul `SYN` al TCP și detectează porturile deschise pe un sistem.

```
razvan@valhalla:~/carte-uso$ nmap -sS localhost
You requested a scan type which requires root privileges.
QUITTING!

root@valhalla:~# nmap -sS localhost

Starting Nmap 4.68 ( http://nmap.org ) at 2009-09-23 15:38 EEST
Interesting ports on localhost (127.0.0.1):
Not shown: 1705 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
25/tcp   open  smtp
53/tcp   open  domain
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
631/tcp  open  ipp
5900/tcp open  vnc
7634/tcp open  hddtemp

Nmap done: 1 IP address (1 host up) scanned in 0.312 seconds
```

Opțiunea `-sS` necesită drepturi privilegiate. În exemplul de mai sus, utilizatorul neprivilegit `razvan` nu a putut rula comanda **`nmap`** cu opțiunea `-sS`, fiind necesară folosirea utilizatorului `root`.

Pentru utilizatorii neprivilegiați, se pot folosi alte opțiuni precum `-sT` (*TCP connect scan*). Comportamentul este similar opțiunii `-sS`, dar mai lent:

```
razvan@valhalla:~/carte-uso$ nmap -sT localhost

Starting Nmap 4.68 ( http://nmap.org ) at 2009-09-23 16:00 EEST
Interesting ports on localhost (127.0.0.1):
Not shown: 1705 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
25/tcp   open  smtp
53/tcp   open  domain
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
631/tcp  open  ipp
5900/tcp open  vnc
7634/tcp open  hddtemp
```

```
17
18 Nmap done: 1 IP address (1 host up) scanned in 0.170 seconds
```

Alte două opțiuni de scanare importante sunt `-sV` și `-O`. Opțiunea `-sV` oferă informații despre versiunea serviciilor ce ascultă pe porturile deschise, în vreme ce opțiunea `-O` oferă informații despre sistemul de operare instalat, precum și timpul de la ultima repornire.

```
1  root@kiwi:~# nmap -sV -O 141.85.37.53
2
3  Starting nmap 3.55 ( http://www.insecure.org/nmap/ ) at 2004-08-27 17:23
   EEST
4  Interesting ports on dhcp-53.cs.pub.ro (141.85.37.53):
5  (The 1659 ports scanned but not shown below are in state: closed)
6  PORT   STATE SERVICE VERSION
7  22/tcp open  ssh     OpenSSH 3.8.1p1 (protocol 1.99)
8  Device type: general purpose
9  Running: Linux 2.4.X|2.5.X|2.6.X
10 OS details: Linux 2.5.25 - 2.6.3 or Gentoo 1.2 Linux 2.4.19 rc1-rc7)
11 Uptime 0.105 days (since Fri Aug 27 14:52:12 2004)
12
13 Nmap run completed -- 1 IP address (1 host up) scanned in 11.840 seconds
```

nmap oferă și alte opțiuni. Comanda de mai jos realizează scanarea porturilor cuprinse între 1 și 100 pe stația locală:

```
1  root@valhalla:~/carte-uso# nmap -p 1-100 localhost
2
3  Starting Nmap 4.68 ( http://nmap.org ) at 2009-09-23 16:14 EEST
4  Interesting ports on localhost (127.0.0.1):
5  Not shown: 96 closed ports
6  PORT   STATE SERVICE
7  21/tcp open  ftp
8  22/tcp open  ssh
9  25/tcp open  smtp
10 53/tcp open  domain
11
12 Nmap done: 1 IP address (1 host up) scanned in 0.107 seconds
```

10.5 Securizarea rețelei

Securizarea unui sistem sau a unei rețele pot presupune investiții importante atât în echipamente, cât și în software. Cu toate acestea, orice soluție de securitate nu trebuie să piardă din vedere câteva componente de bază.

Primul pas în orice soluție de securitate o reprezintă **stabilirea unor politici clare de securitate**. O astfel de politică de securitate va urmări:

- separarea domeniilor cu nivel de securitate diferit;
- definirea clară a drepturilor fiecărui utilizator;
- definirea serviciilor ce trebuie oferite de fiecare componentă a rețelei.

Odată stabilită politica de securitate va trebui restricționat traficul nedorit/nenecesar prin **configurarea politicilor de filtrare a pachetelor** și a unui firewall (vezi secțiunea 10.5.1).

Infectarea stațiilor dintr-o rețea locală poate oferi o poartă de acces unui atacator în spatele firewall-ului, reducând gradul de securitate al rețelei. Din acest motiv un administrator responsabil nu se va ocupa doar de configurarea firewall-ului și a serverelor, ci și de securizarea stațiilor de acces.

O componentă importantă a securității o reprezintă confidențialitatea. În scopul protejării unor date sensibile se recomandă configurarea criptării traficului. Operația de autentificarea trebuie să fie, în general, o operație securizată, așa cum se întâmplă în cazul SSH.

10.5.1 Firewall-uri

Firewall-urile sunt dispozitive sau aplicații care filtrează pachetele de rețea pe baza unor politici sau reguli prestabilite.

Firewall-urile au rolul de protejare a rețelei prin filtrarea traficului nedorit/nenecesar. Un firewall poate fi configurat să filtreze traficul ICMP pentru a preveni atacuri de tip flood, cererile de inițiere de conexiune în rețeaua locală sau orice trafic care nu este menționat explicit ca fiind trafic legitim.

Firewall-urile pot fi dispozitive dedicate (**firewall-uri hardware**) sau aplicații (**firewall-uri software**). Firewall-urile hardware sunt mai rapide, dar prezintă costuri mai mari. Firewall-urile software prezintă avantajul flexibilității, dar nu sunt la fel de eficiente ca firewall-urile hardware.

Aplicația firewall pe sistemele Linux este **iptables**[1]. Exceptând rolul său de filtrare a pachetelor, iptables poate fi folosit și pentru alterarea pachetelor și pentru NAT (*Network Address Translation*).

Sistemele Windows începând cu Windows XP dispun de o aplicație firewall nativă, **Windows Firewall**, dar există numeroase aplicații de tip firewall personal, precum Zone Alarm[2].

10.5.2 Criptarea informației

O tehnică folosită încă din cele mai vechi timpuri, **criptarea** informației înseamnă folosirea unui algoritm care să transforme un mesaj cunoscut într-un mesaj criptat. Aceast algoritm presupune, în general, folosirea unei `chei de criptare`. Operația inversă se cheamă **decriptare**.

Criptarea este folosită pentru a proteja datele de acces neautorizat. Datele protejate pot fi stocate la un moment dat pe un dispozitiv de suport sau pot fi date transmise prin rețea.

În domeniul IT, există două tipuri de criptare: criptarea cu chei simetrice și criptarea cu chei asimetrice.

[1] http://www.netfilter.org/

[2] http://www.zonealarm.com/security/en-us/home.htm?lid=en-us

Criptarea cu chei simetrice presupune folosirea unei chei atât pentru criptare cât și pentru decriptare. Cheia este secretă – este cunoscută doar celor două entități care comunică. Formal, cele două operații sunt descrise astfel:

```
criptare: K(M) = C
decriptare: K(C) = M
```

`K` este cheia, `M` este mesajul inițial, iar `C` este mesajul criptat. Folosirea aceleiași chei peste mesajul criptat duce la obținerea mesajului inițial.

Exemple de algoritmi de criptare cu chei simestrice sunt DES, Twofish, Blowfish, IDEA, AES. AES (*Advanced Encryption Standard*) este algoritmul adoptat oficial de guvernul SUA. Algoritmul a fost ales în urma unui proces de standardizare de 5 ani la care au participat 15 proiecte diferite.

Criptarea cu chei asimestrice folosește o pereche de chei: o cheie publică și o cheie privată. După cum reiese și din denumire, cheia privată este secretă, iar cheia publică este accesibilă de oricine. Există două utilizări pentru o pereche cheie privată/cheie publică:

- **semnături digitale** (*digital signatures*), în care cheia privată este folosită pentru criptarea/semnarea unui document, iar cheia publică este folosită pentru a verifica faptul că documentul este autentic; oricine poate face verificarea dar numai deținătorul chei private poate semna un document;
- **criptare** (*public key encryption*), în care cheia publică este folosită pentru criptarea unui mesaj, iar cheia privată este folosită pentru decriptarea mesajului; oricine poate crea mesajul criptat pentru a fi transmis posesorului cheii private, dar numai acesta din urmă poate decripta mesajul.

Formal, procesele de semnare și verificare și cele de criptare și decriptare sunt descrise mai jos:

```
semnare: p(M) = S
verificare: P(S) = M

criptare: P(M) = C
decriptare: p(C) = M
```

În cazul semnării, se folosește cheia privată `p` pentru a "cripta" mesajul `M` în mesajul `S`. Cheia publică `P` este folosita pentru decriptarea mesajului `S` înapoi în mesajul `M`. În cazul criptării/decriptării, lucrurile se desfășoară asemănător, dar se folosește cheia publică pentru criptare și cheia privată pentru decriptare.

Criptarea cu chei asimetrice este, în general, mai sigură decât criptarea cu chei simetrice: cheia nu trebuie să fie partajată și algoritmii sunt mai robuști. Cu toate acestea, criptarea cu chei asimestrice este mult mai lentă decât criptarea cu chei simetrice. Există situații în care cele două metode se folosesc la un loc. Spre exemplu, în cazul SSH, inițierea conexiunii se realizează folosind criptare cu chei asimetrice; în urma inițierea se realizează o negociere după care se obține o cheie simetrică; această cheie va fi folosită în continuare pentru criptarea comunicației între clientul și serverul SSH.

O extensie a semnăturilor digitale o reprezintă **certificatele digitale**[1]. Un certificat digital conține o cheie publică și un set de informații despre posesorul certificatului semnate digitale. Întrucât cheia publică este inclusă în certificat, oricine poate verifica validitatea informațiilor. Depinzând de infrastructura folosită, semnatura poate fi a unei autorități specializate (CA – *certificate authority*[2], în cazul PKI – *public key infrastructure*[3]) sau a mai multor utilizatori (endorsements), în cazul *web of trust*[4].

10.5.3 Monitorizarea rețelei

Autorii de cărți de rețele de calculatoare sau de ghiduri pentru administratorii de rețea precizează în general că activitatea de monitorizare este una dintre cele mai importante sarcini ale unui administrator. Cu toate acestea, adesea, recomandările propuse sunt deseori limitate la cele evidente precum: "Un administrator trebuie să citească regulat jurnalele de pe fiecare server" sau "Este important să nu uitați să porniți serviciul de monitorizare".

Este important de înțeles că monitorizarea nu se rezumă doar la rularea unui daemon de genul **sysklogd** sau la deschiderea săptămânală a unei aplicații precum **Event Viewer**. Cele două funcții principale ale monitorizării sunt asigurarea dinamică a securității prin alertarea în fața unei încercări de atac, și posibilitatea localizării și definirii unei defecțiuni sau a unei funcționări suboptimale.

Există mai multe criterii pentru a clasifica diversele tipuri de acțiuni de monitorizare. O primă clasificare face distincția între:

- monitorizarea unui sistem specific;
- monitorizarea conectivității;
- monitorizarea traficului din rețea.

Monitorizarea unui dispozitiv specific presupune rularea unui serviciu ce va urmări apelarea celorlalte servicii, precum și accesul la jurnalele create de acest server. Un exemplu este serviciul de jurnalizare oferit de **sysklogd**. **Monit**[5] permite monitorizarea proceselor, fișierelor și poate să execute diverse acțiuni la nevoie.

Monitorizarea conectivității se bazează deseori pe mecanisme foarte simple, cel mai important dintre acestea fiind folosirea utilitarului `ping`, metoda fiind denumită **ping sweep**. Crearea unui sistem de interogări ICMP periodice poate oferi administratorului informații despre întreruperea unui segment de rețea sau despre defectarea sau stingerea unuia dintre servere. Un astfel de sistem de interogări poate fi realizat ori printr-un script simplu, ori se poate apela la aplicații dedicate.

Monitorizarea de rețea cuprinde numeroase componente, de la dispozitive hardware dedicate (denumite generic **IDS** – *Intrusion Detection System*), la programe de interceptare a traficului în rețea (programe de sniffing), precum și programe de evaluare

[1] http://en.wikipedia.org/wiki/Digital_certificate
[2] http://en.wikipedia.org/wiki/Certificate_authority
[3] http://en.wikipedia.org/wiki/Public_key_infrastructure
[4] http://en.wikipedia.org/wiki/Web_of_trust
[5] http://mmonit.com/monit/

a securității unui sistem din rețea, în această categorie intrând programele de scanat porturi. O aplicație foarte cunoscută de monitorizare a rețelei este **Nagios**[1].

Apariția switchurilor în rețelele locale a transformat mediul partajat al Ethernetului într-un mediu dedicat. Cu alte cuvinte, într-o rețea cu switchuri, pachetele schimbate între două stații nu vor mai fi vizibile de o a treia stație aflată în aceeași rețea locală. În această situație, un administrator de rețea ce dorește monitorizarea traficului către o anumită destinație va trebui ca înainte de a lansa programul de interceptare a traficului să facă un atac **ARP poisoning**[2] pentru respectiva destinație. Pentru aceasta se pot folosi utilitare precum Cain[3] și dsniff[4].

Utilitarul tcpdump

tcpdump este un utilitar destinat monitorizării traficului în rețea și achiziționării de date.

Plăcile de rețea, cum sunt cele Ethernet, capturează la nivelul legătură de date doar cadrele adresate lor sau pe cele de difuzare. De aceea, pentru a captura toate cadrele, `tcpdump` trebuie să treacă interfața într-un mod special de lucru, numit *promiscuous mode*, pentru care are nevoie de suportul sistemului de operare.

La o rulare simplă, `tcpdump` capturează pachetele de pe prima interfață de rețea a sistemului și afișează informații din antetul IP și TCP.

```
1 root@frodo:~# tcpdump
2 tcpdump: listening on eth0
3 14:45:09.252803 frodo.noi.39993 > 64.12.30.90.5190: P
  444608787:444608793(6) ack 1453900500 win 32893 (DF)
4 14:45:09.254097 frodo.noi.33335 > main.noi.domain: 16713+ PTR?
  90.30.12.64.in-addr.arpa.(42) (DF)
5 14:45:09.254583 main.noi.domain > frodo.noi.33335:  16713 NXDomain 0/1/0
  (113)(DF)
```

În exemplul de mai sus, fiecare linie conține:

- timpul în care pachetul TCP a intrat în rețeaua locală;
- informații legate de antetul IP (adresa sursă și adresa destinație);
- informații din antetul TCP (port sursă, pot destinație, numere de secvență, flag-uri TCP).

Pentru a vedea și conținutul pachetului se poate utiliza opțiunea `-x`:

```
1 root@frodo:~# tcpdump -c1 -x
2 tcpdump: listening on eth0
3 18:38:51.087244 64.12.24.48.5190 > frodo.noi.32777: P
  3127656695:3127656809(114) ack 3009470569 win 16384 (DF) [tos 0x80]
4                          4580 009a c7a1 4000 6406 354d 400c 1830
5                          c0a8 000b 1446 8009 ba6c 40f7 b360 e069
6                          5018 4000 1f0d 0000 2a02 20dc 006c 0003
7                          000b 0000 87d9 0d61 0836 3032 3033 3631
8                          3400 0000 0700 0100 0200 7000 0c00 2500
```

[1] http://www.nagios.org/
[2] http://en.wikipedia.org/wiki/ARP_spoofing
[3] http://www.oxid.it/cain.html
[4] http://www.monkey.org/~dugsong/dsniff/

```
9                          0000
10 6 packets received by filter
11 0 packets dropped by kernel
```

Conform structurii unui pachet IP, primii 20 octeți (4580 009a c7a1 4000 6406 354d 400c 1830 c0a8 000b) reprezintă antetul IP iar următorii 20 de octeți (1446 8009 ba6c 40f7 b360 e069 5018 4000 1f0d 0000) reprezintă antetul TCP.

Există soluții grafice ce realizează interpretarea conținutului pachetului. Cea mai cunoscută aplicație grafică de interceptare și interpretare a traficului de rețea este **Wireshark**[1], utilitar disponibil atât în mediul Windows, cât și în Linux. Echivalentul pentru `tcpdump` în lumea Windows este `windump`[2].

10.6 Studiu de caz

10.6.1 Drepturile pe fișiere în NTFS

Sistemul de fișiere NTFS oferă o serie de îmbunătățiri față de FAT, atât din punctul de vedere al performanței, cât și al securității. Într-un sistem de fișiere FAT, toate conturile de utilizator au drepturi egale și depline asupra sistemului de fișiere.

Figura 10.2: Precizarea drepturilor în NTFS

Securitatea în NTFS se implementează prin intermediul mecanismului de liste de acces. O listă de acces reprezintă un set descriptori atașati fișierelor și directoarelor din sistemul

[1] http://www.wireshark.org/
[2] http://www.winpcap.org/windump/install/

de fișiere. Pentru ca un proces să poată accesa un fișier sau un director, sistemul de securitate verifică dacă procesul are dreptul de a-l accesa.

Ca măsură sporită de securitate, sistemul de fișiere NTFS oferă posibilitatea criptării datelor pe care le stochează. Avantajul implementării mecanismelor de criptare direct în sistemul de fișiere îl reprezintă viabilitatea pe care acest tip de securitate o menține în lucrul cu orice sistem de operare care oferă suport pentru sistemul respectiv de fișiere.

Din mediul Windows, criptarea unui fișier sau a unui director (inclusiv a conținutului său) se face din ecranul de proprietăți al fișierului/directorului (clic dreapta > Properties > Advanced > "Encrypt contents to secure data").

Pentru a putea acționa granular asupra securității sistemului de fișiere, în Windows trebuie dezactivată opțiunea "Use simple file sharing" (de la Explorer > Folder Options > View > Use Simple File Sharing: debifare). Ulterior, atribuirea drepturilor se face din fereastra de proprietăți a unui director sau a unui fișier, sub tabul `Security`. Se selectează utilizatorul sau grupul care va fi editat, iar din lista de drepturi se indică permisiunile ce vor fi atribuite. Toate configurările se aplică doar fișierului sau directorului selectat. Pentru a configura explicit permisiunile, butonul `Advanced` afișează o nouă interfață în care pot fi vizualizate toate tipurile de permisiuni ce sunt aplicate fișierului sau directorului ales, inclusiv drepturi moștenite prin ierarhia sistemului de fișiere.

10.6.2 Recuperarea parolei

Un bun administrator de sistem trebuie să fie conștient de mecanismele de recuperare a parolei pentru a putea preveni proceduri neautorizate de modificare a acesteia.

Există două metode importante pentru a realiza recuperarea parolei de root, ambele necesitând accesul fizic la echipament. Prima metodă presupune modificarea din meniul bootloader-ului a parametrilor trimiși la încărcarea nucleului, iar a doua se bazează pe folosirea unui CD Linux bootabil.

Editarea GRUB

Prima metodă de a parolei de root necesită editrea meniului de boot a GRUB. Din meniul GRUB se selectează imaginea de nucleu ce se dorește încărcată și se activează editarea respectivei linii de grub prin opțiunea `e`. În capitolul 6 sunt prezentate pe larg metodele de modificare a opțiunilor transmise la încărcarea nucleului.

La linia conținând imaginea de nucleu selectată se adaugă opțiunea `init=/bin/bash` și se continuă procesul de inițializare a sistemului. În urma încheierii procesului de inițializare se va obține un shell root. Dacă la execuția comenzii **`passwd`** se primește mesaj de eroare trebuie remontat sistemul de fișiere pentru scriere și citire. Acest lucru se realizează folosind comanda:

```
1 root@ubuntu:~# mount -o remount,rw /
```

Pentru a preveni o recuperare neautorizată a parolei de root se recomandă folosirea parolelor de acces la meniul bootloader-ului. În cazul GRUB, acest lucru se realizează cu ajutorul opțiunii `password` în cadrul fișierului de configurare a GRUB.

Folosirea unui Live CD

A doua metodă de recuperare a parolei presupune folosirea unui Live CD Linux. Dacă ordinea de boot nu acordă prioritate unității optice în fața hard discului, va trebui schimbată ordinea de boot din BIOS.

După inițializarea sistemului de pe CD trebuie obținut un shell de root. În cazul Live CD-ului Ubuntu aceasta se realizează prin executarea din contul neprivilegiat a comenzi **sudo**:

```
1 root@ubuntu:~# sudo bash
```

Trebuie creat un nou director în care se va face montarea vechiului sistem de fișiere:

```
1 root@ubuntu:~# mkdir /mnt/x
2
3 root@ubuntu:~# mount /dev/hda2 /mnt/x
```

În acest moment putem invoca comanda chroot:

```
1 root@ubuntu:~# chroot /mnt/x
```

Ultimul pas presupune invocarea utilitarului **passwd** și schimbarea parolei de root. Întrucât, prin intermediul Live CD-ului, utilizatorul are acces privilegiat la întreg sistemul, va avea acces complet la sistemul de fișiere de pe discul sistemului. Invocarea comenzii **passwd** va conduce la modificarea informației din fișierul `/etc/shadow`.

Pentru a preveni o astfel de procedură este necesar să se forțeze din BIOS ca prim dispozitiv de boot unul dintre hard discuri și să se definească o parolă de acces la meniul de configurare a BIOS-ului.

Cuvinte cheie

- cel mai mic privilegiu
- cea mai slabă verigă
- controlul accesului
- autentificare
- autorizare
- criptare
- atacuri
- chmod
- chown
- chroot
- df
- DoS
- drepturi pe fișiere în NTFS
- drepturi pe fișiere în UNIX
- du
- /etc/sudoers
- /etc/passwd, /etc/shadow
- /etc/security/limits.conf
- liste de control a accesului – ACL
- matrice de drepturi
- firewall
- criptare, decriptare
- chei simetrice, chei asimetrice
- chei private, chei publice
- semnătură digitală, certificat digital
- hacker

- netstat
- monitorizare
- nmap
- parole
- politică de securitate
- quota
- securitatea fizică
- sniffer
- sudo
- suid
- syslog
- tcpdump
- top
- ulimit
- umask
- viruși

Întrebări

1. Care dintre următoarele NU este o funcționalitate oferită de utilitarul **`nmap`**?
   - ❑ scanarea porturilor UDP și TCP
   - ❑ detectarea versiunii sistemului de operare
   - ❑ detectarea stațiilor active din rețeaua locală
   - ❑ filtrarea pachetelor de rețea
2. Fie următoarele comenzi:

```
1 umask 022
2 echo "Acesta nu este un test" > test
3 chmod go+x test
```

   Care vor fi drepturile pentru utilizatorii grupului fișierului `test`?
   - ❑ doar citire
   - ❑ doar execuție
   - ❑ citire și execuție
   - ❑ citire, scriere și execuție
3. Care dintre parolele de mai jos este cea mai sigură pentru utilizatorul `cornel`?
   - ❑ cornelush
   - ❑ coRnel
   - ❑ _cOrN31[]sh+
   - ❑ niciuna, toate parolele vor fi încercate într-un atac bazat pe dicționar
4. Care dintre riscurile de mai jos reprezintă riscuri de securitate fizică pentru un server?
   - ❑ lipsa unui firewall
   - ❑ parole pentru utilizatori prea simple
   - ❑ vârfuri de tensiune în rețeaua electrică

- ❑ absența licenței pentru sistemul de operare

5. În urma procesului de scanare de porturi se poate determina:
   - ❑ dacă un serviciu rulează pe un port dat
   - ❑ versiunea serviciului
   - ❑ dacă serviciul este filtrat
   - ❑ toate variantele
6. Primirea unui email ce conține un virus pe un sistem ce nu are antivirus duce automat la infectarea sa. Un sistem infectat cu un virus nu mai poate fi țința unui alt atac.
   - ❑ adevărat, adevărat
   - ❑ adevărat, fals
   - ❑ fals, adevărat
   - ❑ fals, fals
7. Traficul către un server este monitorizat cu **`tcpdump`**. Care dintre următoarele informații NU poate obținută?
   - ❑ portul pe care rulează serviciul
   - ❑ numărul de clienți ai server-ului
   - ❑ dimensiunea pachetelor folosite
   - ❑ adresa IP a serverului
8. Pentru a limita numărul de procese per utilizator se poate folosi:
   - ❑ **`ulimit`**
   - ❑ **`quota`**
   - ❑ **`nmap`**
   - ❑ **`chroot`**
9. Care este primul pas în securizarea unei rețele?
   - ❑ definirea unei politici de securitate
   - ❑ definirea regulilor de filtrare pe firewall
   - ❑ instalarea de programe antivirus pe toate stațiile din rețea
   - ❑ criptarea traficului
10. Pe un sistem Linux, ce fișier stochează parolele utilizatorului (criptate)?
    - ❑ `/etc/passwd`
    - ❑ `/etc/shadow`
    - ❑ `/etc/security/limits.conf`
    - ❑ `/var/log/apache2/access.log`

Capitolul 11

Compilare și linking

He who hasn't hacked assembly language as a youth
has no heart. He who does as an adult has no brain.
John Moore

Ce se învață din acest capitol?

- Cum se obține un program executabil dintr-un program sursă
- Compilarea și interpretarea unui program
- Etapele compilării
- Utilizarea GCC pentru compilarea și link-editarea unui program
- Optimizarea compilării
- Fișiere obiect și fișiere executabile
- Biblioteci de funcții
- Automatizarea acțiunilor cunoscând dependențele între ele; utilitarul make
- Concepte de portabilitate
- Link-editarea codului C și C++

11.1 Introducere

Unul dintre aspectele importante în utilizarea unui sistem de operare este dezvoltarea de noi programe. Acest lucru presupune folosirea unui limbaj de programare pentru scrierea de cod sursă. Codul sursă este compilat, iar programul executabil obținut este rulat. Acțiunea de scriere de cod sursă poartă numele de programare sau dezvoltare. Persoana care scrie codul se numește programator sau dezvoltator. Termenii în limba engleză sunt **programmer**, **software developer** sau **software engineer**.

Sistemele de operare moderne pun la dispoziția dezvoltatorului/programatorului o suită de aplicații folosite pentru scrierea de noi programe. În cazul familiei de sisteme de operare Windows, suita de aplicații Visual Studio reprezintă forma cea mai cunoscută

pentru dezvoltarea de noi aplicații. Aceasta permite crearea de programe în diverse limbaje de programare și cu utilitate diversă. Dezvoltarea de aplicații se desfășoară într-un mediu integrat (*IDE* – Integrated Development Environment) (vezi secțiunea 14.9).

De partea cealaltă, sistemele Linux pun la dispoziția utilizatorului suita de aplicații GNU, folosită pentru compilarea, link-editarea și depanarea programelor. În mod tradițional, dezvoltatorii de aplicații pe sisteme Linux folosesc utilitare dedicate (editor, compilator, depanator), dar se folosesc din ce în ce mai mult și mediile integrate precum Eclipse sau Anjuta (vezi secțiunea 14.9.1 și secțiunea 14.9.2).

11.1.1 Editoare

Prima fază a procesului de dezvoltare este scrierea de cod specific unui limbaj de programare. Codul reprezintă o înșiruire de instrucțiuni, variabile și funcții pentru acel limbaj. Limbajul impune o sintaxă pe care programatorul trebuie să o respecte. Fișierul în care se scrie codul este, în general, un fișier text și poartă numele de fișier sursă, sau mai simplu sursă. Expresia "trebuie să mă uit prin surse"[1] se referă la parcurgerea fișierelor sursă ale unei aplicații pentru analiza codului.

Scrierea de cod sursă se realizează prin intermediul unui editor. Editoare răspândite în lumea Unix sunt vi, Emacs, nano, joe, kate, gedit. Editoare răspândite în lumea Windows sunt Notepad++, Ultraedit, Crimson Editor. De obicei, în lumea Windows, editoarele sunt parte a unui IDE (Integrated Development Environment) cum este Visual Studio sau Eclipse.

Un editor este folosit pentru scrierea unui fișier sursă. Drept urmare, editoarele moderne prezintă o serie de caracteristici utile acestui scop: indentare automata, colorare a codului (syntax highlighting), code folding, variable completion, folosirea de registre de editare pentru operatii multiple etc.

Lumea Unix este dominată de rivalitatea dintre editoarele vi și Emacs[2]. În vreme ce vi este un adept al filozifiei Unix (keep it simple) și este folosit doar pentru editarea de fișiere sursă, Emacs este un editor complex cu facilități care includ posibilitatea de folosire ca mediu integrat de dezvoltare (IDE), client de e-mail, client de IRC etc. De-a lungul vremii, însă, numeroase facilități au fost adăugate și editorului vi astfel încât funcționalitățile oferite sunt apropiate. vi și Emacs reprezintă de fapt două familii de editoare. Cei mai cunoscuți reprezentanți ai celor două familii sunt, respectiv, editorul Vim[3] și GNU Emacs[4]. Ambele editoare dispun de facilități avansate pentru editare eficientă și permit posibilitatea de extensie și personalizare. Cunoscuți pentru fanatismul folosirii unuia dintre cele două editoare, programatorii vor prezenta fișierul propriu de configurare (.vimrc în cazul Vim și .emacs în cazul GNU Emacs).

O prezentare mai amplă a editorului Vim găsiți în capitolul 14.

---

[1] http://encyclopedia2.thefreedictionary.com/Use+the+Source+Luke

[2] http://en.wikipedia.org/wiki/Editor_war

[3] http://vim.sourceforge.net/

[4] http://www.gnu.org/software/emacs/

11.1.2 Compilare și interpretare

Programul scris în cod sursă este compilat sau interpretat pentru a îndeplini sarcina pentru care a fost scris. Atât compilarea cât și interpretarea unui program sunt forme de translatare a acestuia și execuție pe un sistem hardware. Translatarea se referă la transformarea într-un limbaj a unui program scris într-un alt limbaj.

În definiția cea mai simplă, compilarea unui program se referă la obținerea unui fișier binar executabil. Executarea acestui fișier duce la îndeplinirea sarcinii pentru care a fost scris programul. După cum se va observa în secțiunea 11.3 există mai multe faze specifice compilării unui program.

De partea cealaltă, interpretarea unui program elimină faza obținerii unui executabil și execută codul asociat unui program pe măsura parcurgerii acestuia. Avantajul interpretării unui program este timpul mai scurt de dezvoltare (se elimină faza de compilare) și depanarea mult mai facilă. Pe de altă parte, dezavantajul interpretării unui program este viteza mai scăzută de execuție. În cazul compilării, rularea executabilului înseamnă rularea de cod specific sistemului fizic. În cazul interpretării, codul sursă este parcurs, analizat și apoi executat, rezultând într-un timp de rulare mai mare.

Limbaje compilate și limbaje interpretate

Utilitarul folosit pentru compilarea unui program poartă numele de compilator, iar cel folosit pentru interpretare poartă numele de interpretor. Compilatorul, respectiv interpretorul, sunt **implementări** ale limbajului de programare asociat. Astfel, compilatorul/interepretorul cunoaște sintaxa limbajului de programare în care a fost scris respectivul program și este folosit pentru translatarea acestuia într-o formă executabilă: fișier executabil în cazul compilatorului, sau execuție în timpul translatării în cazul interpretorului.

Una dintre cele confuziile din lumea limbajelor de programare și a programării în general este clasificarea în limbaje compilate și limbaje interpretate. De multe ori se menționează că C, C++, Ada, Fortran, Java sunt limbaje compilate, în vreme ce Perl, Python, PHP, Ruby, Lisp sunt limbaje interpretate.

De fapt, orice limbaj poate fi atât interpretat cât și compilat. Astfel, deși limbajul C este de obicei compilat, există și interpretoare pentru acesta[1].

În concluzie, nu există limbaje interpretate sau compilate ci **implementări de compilator** sau **implementări de interpretor** pentru un limbaj. Într-adevăr, limbajele au asociate preponderent un anumit tip de implementare (C are implementări de compilator, Perl are implementări de interpretor) și, drept urmare, sunt denumite "limbaje compilate" sau "limbaje interpretate". Totuși, această clasificare este forțată și recomandăm să fie atribuită **implementării** acelui limbaj.

[1] http://www.softintegration.com/

Limbaje de nivel scăzut și limbaje de nivel înalt

Dintre multiplele clasificări posibile ale limbajelor de programare[1], o clasificare răspândită este în limbaje de programare de nivel scăzut și limbaje de programare de nivel înalt.

> **Limbajele de programare de nivel scăzut** sunt acele limbaje care nu oferă o abstractizare a arhitecturii sistemului de calcul. În mod obișnuit, aceste limbaje oferă o translatare minimă a instrucțiunilor arhitecturii.

Limbaje de programare de nivel scăzut sunt codul mașină, adică scrierea unui program direct în binar și limbajul de asamblare. Un programator nu va scrie niciodată cod mașină și foarte rar cod în limbaj de asamblare.

> **Limbajele de programare de nivel înalt** sunt limbaje de programare care ofera o abstractizare a arhitecturii sistemului de calcul. Limbajele de programare de nivel înalt sunt mai ușor portabile și oferă o sintaxă mai apropiată de limbajul natural.

Limbajele de programare de nivel înalt oferă o plajă largă de abstractizări. Unele limbaje pot ascunde complet detalii legate de modul de adresare a procesorului, alocarea/dezalocarea memoriei, lucrul cu obiecte etc. Noțiunea de limbaj de nivel înalt devine relativă raportat la facilitățile oferite. Dacă în anii '70, limbajul de programare C era considerat limbaj de nivel înalt, în zilele noastre poate fi considerat limbaj de nivel scăzut în comparație cu limbaje precum Java, Python, Ruby.

Exemple de limbaje de nivel înalt sunt: C/C++, Java, Perl, Python, PHP, Ruby, Lisp, Scheme, Prolog, Haskell.

11.1.3 De la sursă la executabil, de la executabil la proces

După cum s-a precizat, în urma compilării unui fișier sursă se obține un executabil care poate fi apoi rulat pentru îndeplinirea sarcinii pentru care a fost scris. În capitolul 5 s-a menționat că un executabil este o entitate pasivă care descrie modul în care va rula programul. Rularea efectivă se realizează în urma execuției programului în cadrul unui proces. Procesul este entitatea activă în cadrul căreia se vor aloca resursele programului și se va rula codul asociat pe procesor.

Așa cum reiese și din figura 11.1, obținerea unui executabil dintr-un fișier sursă se realizează în urma compilării acestuia din urmă. La fel, procesul este obținut din executabil în urma execuției. Utilizatorul va trebui să cunoască doar limbajul de nivel înalt și să scrie codul sursă în acest limbaj. În continuare, compilatorul va obține executabilul; executabilul este un fișier binar care descrie, în cod mașină, operațiile precizate de utilizator. În faza de execuție, cu suportul nucleului sistemului de operare, din executabilul obținut anterior se creează un proces. Procesul este entitatea activă care va executa pe procesor operațiile descrise în cod mașină în zona de cod a executabilului (vezi secțiunea 5.1.2).

[1] http://en.wikipedia.org/wiki/Categorical_list_of_programming_languages

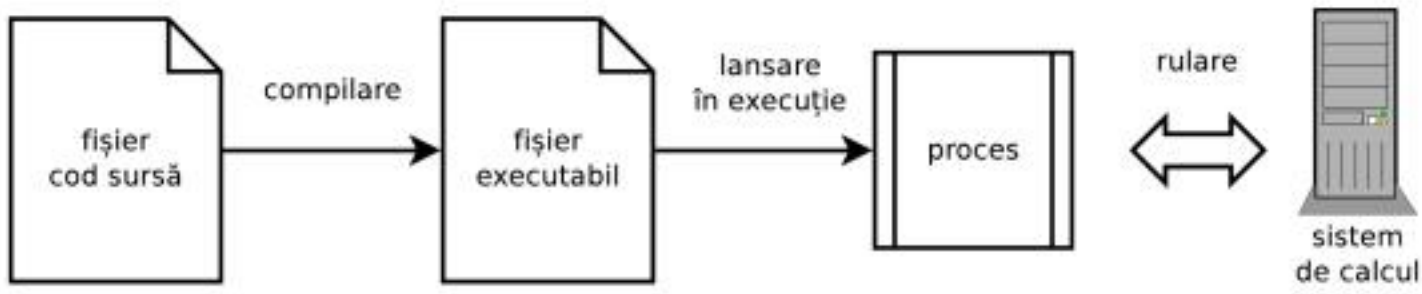


Figura 11.1: De la fișier sursă la executabil; de la executabil la proces

Compilatorul oferă în acest fel un beneficiu important: utilizatorul nu trebuie să dețină informații despre sistemului fizic din spate (zone de memorie, instrucțiuni ale procesorului etc.), ci numai noțiuni despre limbajul folosit pentru a scrie codul sursă. Compilatorul va fi responsabil cu translatarea codului sursă (fișierul sursă, text) în cod mașină (fișierul executabil, binar). Ulterior, în urma rulării executabilului, sistemul de operare creează procesul care execută codul pe procesor.

Folosirea unui compilator aduce după sine avantajul portabilității codului: utilizatorul va scrie cod sursă portabil specific limbajului de programare. Compilatorul generează cod binar neportabil specific sistemului fizic pe care acesta va rula. Mai multe detalii despre portabilitate sunt prezentate în secțiunea 11.6.

În exemplul de mai jos este prezentat un fișier sursă C (`hw.c`) și comenzile pentru compilarea acestuia (**`gcc hw.c`**), generarea unui executabil (`a.out`) și executarea acestuia în cadrul unui proces (**`./a.out`**).

```c
#include <stdio.h>

int main(void)
{
        printf("Hello, World!\n");

        return 0;
}
```

Listing 11.1: hw.c

```
mircea@cougar:~/carte-uso/cap-10$ file hw.c
hw.c: ASCII C program text

mircea@cougar:~/carte-uso/cap-10$ gcc hw.c

mircea@cougar:~/carte-uso/cap-10$ file a.out
a.out: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/
Linux 2.4.1, dynamically linked (uses shared libs), for GNU/Linux 2.4.1,
not stripped

mircea@cougar:~/carte-uso/cap-10$ ./a.out
Hello, World!
```

11.1.4 Pachete necesare

Pentru dezvoltarea de aplicații este nevoie de

- cunoștințe despre un anumit limbaj de programare;

- un editor folosit pentru a scrie codul sursă;
- un interpretor sau un compilator pentru acel limbaj.

În restul acestui capitol, limbajul de programare folosit pentru exemplificare va fi C. Compilatorul folosit va fi GCC[1], compilatorul implicit pe distribuțiile Linux. Pachetele necesare pentru dezvoltarea de aplicații C sub Ubuntu sunt `gcc`, `libc6-dev`. Se recomandă și paginile de manual pentru dezvoltare din pachetul `manpages-dev` și documentația bibliotecii standard C din pachetul `glibc-doc` (sau `glibc-doc-reference` pe sistemele Debian). Instalarea utilitarului `Make` (vezi secțiunea 11.5) este, de asemenea, recomandată.

```
1 root@cougar:~# apt-get install gcc libc6-dev manpages-dev glibc-doc make
```

Mai ușor, se poate instala pachetul virtual `build-essential`. Destinat formal creării de pachete .deb, instalarea acestui pachet conduce la instalarea pachetelor importante pentru dezvoltarea programelor pe un sistem Ubuntu. Pachetul nu include pachetele de documentație, deci acestea vor trebui instalate separat:

```
1 root@cougar:~# apt-get install build-essential manpages-dev glibc-doc
```

11.2 Compilare. GCC

În general, procesul de compilare se referă la obținerea unui fișier executabil dintr-un fișier cod sursă. Aplicația care realizează această translatare se numește compilator. Un compilator este o aplicație complexă care, pe lângă rolul de translator, trebuie să îndeplinească și alte cerințe. Codul binar obținut de compilator în cadrul executabilului va trebui să satisfacă una (sau mai multe) din următoarele solicitări:

- să fie cât mai mic;
- să ruleze cât mai rapid;
- să consume cât mai puțină memorie în momentul rulării.

Un compilator trebuie să implementeze cât mai mult din specificațiile standardelor în rigoare. Spre exemplu, în cazul C, standardele sunt stabilite de ISO[2]. Ultimul standard este ISO 9899:1999 denumit și C99[3].

Pe sistemele Unix, compilarea se realizează prin transmiterea fișierului sursă ca argument compilatorului. Compilarea rezultă în generarea executabilului implicit `a.out`[4]:

```
1 mircea@cougar:~/carte-uso/cap-10$ gcc hw.c
2
3 mircea@cougar:~/carte-uso/cap-10$ ls
4 a.out hw.c
5
6 mircea@cougar:~/carte-uso/cap-10$ ./a.out
7 Hello, World!
```

[1] http://gcc.gnu.org/
[2] http://www.iso.org/iso/home.htm
[3] http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1256.pdf
[4] http://en.wikipedia.org/wiki/A.out

Pe sistemele Windows compilarea se realizează, în linia de comandă, folosind utilitarul **cl**[1]. Executabilul obținut are același nume cu fișierul sursă însă cu extensia `.exe`:

```
C:\> cl hw.c

C:\> hw.exe
Hello, World!
```

GCC este compilatorul implicit pe distribuțiile Linux și pe un număr mare de alte sisteme Unix. Scris de Richard Stallman[2] în 1989 pentru proiectul GNU, era folosit inițial doar pentru a compila programe C (GNU C Compiler). De-a lungul timpului s-a extins, și în ziua de azi, poate compila programe scrise în C++, Fortran, Objective-C, Pascal, Java etc. Drept urmare, denumirea sa a fost schimbată în GNU Compiler Collection. Pachetul GCC oferă executabile (denumite și front-end-uri) pentru fiecare tip de limbaj pe care îl poate compila:

- **gcc** este folosit pentru a compila programe scrise în C;
- **g++** compilează programe scrise în C++;
- **gnat** compilează programe scrise în Ada;
- **gcj** compilează programe scrise în Java;
- **g77** compilează programe scrise în Fortran;

11.2.1 Utilizare GCC

La o rulare implicită a GCC, acesta generează executabilul `a.out`. Fie următorul program C:

```
#include <stdio.h>

main()
{
        printf("Hello, World!\n");
}
```

Listing 11.2: hw2.c

În forma sa cea mai simplă, compilarea se realizează prin transmiterea fișierului sursă ca argument comenzii **gcc**:

```
mircea@cougar:~/carte-uso/cap-10$ gcc hw2.c
```

În mod implicit se obține executabilul `a.out`. Pentru rularea acestuia se folosește construcția `./` care precizează execuția din directorul curent:

```
mircea@cougar:~/carte-uso/cap-10$ ./a.out
Hello, World!
```

Se poate specifica numele executabilului care se dorește obținut prin folosirea opțiunii `-o` urmată de numele executabilului:

[1] http://msdn.microsoft.com/en-us/library/ms235639(VS.80).aspx
[2] http://en.wikipedia.org/wiki/Richard_Stallman

```
mircea@cougar:~/carte-uso/cap-10$ gcc hw2.c -o my_exec

mircea@cougar:~/carte-uso/cap-10$ ./my_exec
Hello, World!
```

În general, executabilele în Unix nu au extensie. Opțiunea `-o` poate fi prezentă oriunde în linia de comandă, doar că va trebui să fie urmată de numele executabilului:

```
mircea@cougar:~/carte-uso/cap-10$ gcc -o my_other_exec hw2.c

mircea@cougar:~/carte-uso/cap-10$ ./my_other_exec
Hello, World!
```

O greșeală frecventă este folosirea opțiunii `-o` urmată imediat de numele fișierului sursă:

```
mircea@cougar:~/carte-uso/cap-10$ gcc -o hw2.c my_other_exec
my_other_exec:(.rodata+0x0): multiple definition of `_fp_hw'
/usr/lib/gcc/i486-linux-gnu/4.1.2/../../../../lib/crt1.o:(.rodata+0x0):
first defined here
[...]
```

ATENȚIE: Această greșeală are consecințe grave pentru că duce la pierderea fișierului sursă.

Programul `hw2.c` prezentat mai sus compilează, dar nu respectă întru totul standardul C. De exemplu, funcția `main` trebuie să întoarcă `int`[1]. Compilatorul nu va afișa, în mod implicit, niciun fel de avertisment pentru astfel de inconsistențe. Acest lucru poate fi schimbat prin folosirea opțiunii `-Wall` (*Warnings All*) care comandă compilatorul să afișeze mesaje de avertizare pentru abateri de la standard sau pentru formulări atipice:

```
mircea@cougar:~/carte-uso/cap-10$ gcc -Wall hw2.c -o my_exec
hw2.c:4: warning: return type defaults to 'int'
hw2.c: In function 'main': hw2.c:6: warning: control reaches end of non-
void function
```

După corectare, avertismentele nu vor mai fi furnizate:

```c
#include <stdio.h>

int main(void)
{
        printf("Hello, World!\n");

        return 0;
}
```

Listing 11.3: hw2.c

```
mircea@cougar:~/carte-uso/cap-10$ gcc -Wall hw2.c -o my_exec

mircea@cougar:~/carte-uso/cap-10$
```

O greșeală frecventă care poate fi eliminată prin folosirea opțiunii `-Wall` este utilizarea construcției `if (a = 1)`. Programul de mai jos folosește o astfel de construcție care este raportată, însă, de compilator în momentul folosirii opțiunii `-Wall`:

[1]Una dintre greșelile frecvente realizate de programatorii începători este folosirea formelor **void** `main(`**void**`)` sau `main()`. Standardul C specifică forma **int** `main(`**void**`)` ca fiind cea corectă. Nefolosirea acestei forme poate conduce la probleme - http://users.aber.ac.uk/auj/voidmain.shtml

```c
#include <stdio.h>

int main(void)
{
        int a = 1;

        if (a = 1)
                printf("Hello, World!\n");

        return 0;
}
```

Listing 11.4: hw3.c

```
mircea@cougar:~/carte-uso/cap-10$ gcc -Wall hw2.c -o my_exec
hw2.c: In function 'main': hw2.c:7: warning: suggest parentheses around
assignment used as truth value
```

IMPORTANT: Din cauza sintaxei relaxate a limbajului C este recomandată folosirea opțiunii `-Wall` la orice compilare de program.

11.2.2 Compilarea din surse multiple

O aplicație complexă va avea mai multe fișiere sursă conținând diverse părți din implementare. În exemplul de mai jos, fișierul `main.c` conține funcția `main`, fișierele `add.c` și `sub.c` conțin, respectiv, implementarea funcțiilor `add` și `sub`, iar fișierul `func.h` deține antetele (declarațiile) acelor funcții.

```c
#include <stdio.h>

#include "func.h"

#define NUM     10

int main(void)
{
        printf("Suma(%d) = %d\n", NUM, sum(NUM));
        printf("Suma-alternanta(%d) = %d\n", NUM, sumalt(NUM));

        return 0;
}
```

Listing 11.5: main.c

```c
#ifndef FUNC_H_
#define FUNC_H_         1

int sum(int n);
int sumalt(int n);

#endif
```

Listing 11.6: func.h

```c
#include "func.h"

```

```
3  int sum(int n)
4  {
5          int i, sum;
6
7          sum = 0;
8          for (i = 1; i <= n; i++)
9                  sum += i;
10
11         return sum;
12 }
```

Listing 11.7: sum.c

```
1  #include "func.h"
2
3  int sumalt(int n)
4  {
5          int i, sum;
6
7          sum = 0;
8          for (i = 1; i <= n; i++) {
9                  if (i % 2 == 0)
10                         sum -= i;
11                 else
12                         sum += i;
13         }
14
15         return sum;
16 }
```

Listing 11.8: sumalt.c

Pentru obținerea unui executabil, toate aceste fișiere vor fi transmise ca argument compilatorului:

```
1  razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ gcc -Wall main.c
   sum.c sumalt.c
2
3  razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ ./a.out
4  Suma(10) = 55
5  Suma-alternanta(10) = -5
```

Se poate folosi opțiunea `-o` pentru a specifica numele executabilului final:

```
1  razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ gcc -Wall main.c
   sum.c sumalt.c -o exec
2
3  razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ ./exec
4  Suma(10) = 55
5  Suma-alternanta(10) = -5
```

11.3 Etapele compilării (inclusiv link-editarea)

Exemplele de până acum au folosit GCC pentru a obține dintr-un fișier sau mai multe fișiere sursă un executabil. Acest lucru a fost realizat într-o singură comandă. Totuși, rularea unei instanțe a GCC presupune trecerea printr-un set de etape de prelucrare a fișierului sursă până la executabil. Aceste faze sunt, în ordine:

1. preprocesarea;
2. compilarea;
3. asamblarea;
4. link-editarea.

O rulare simplă a comenzii **gcc** înseamnă trecerea prin toate aceste faze. Prezența unor parametri poate forța GCC să se oprească după una dintre ele.

Figura 11.2 sintetizează etapele compilării:

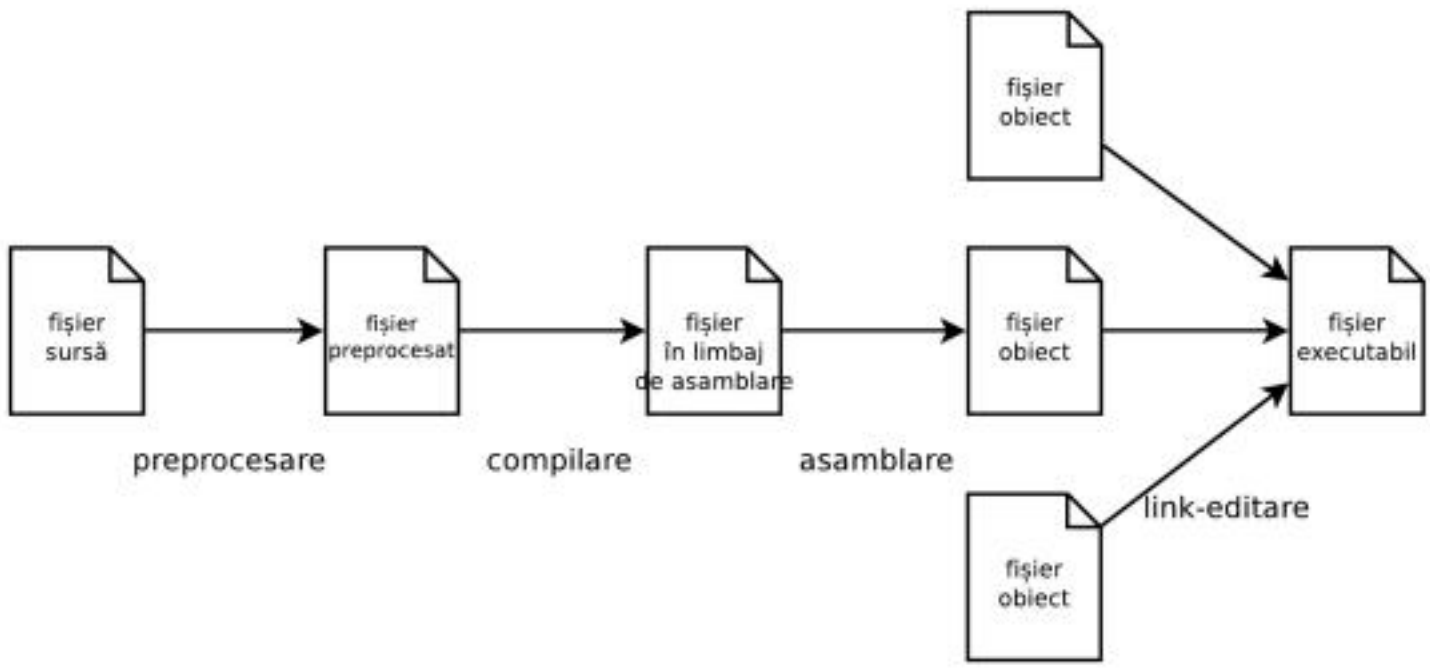


Figura 11.2: Fazele compilării

În cazul folosirii directe a comenzii **gcc**, se realizează integrat toate cele 4 faze. Fișierele obținute în urma fiecărei faze sunt, în această situație, stocate temporar în sistemul de fișiere după care sunt șterse. O rulare de tip *verbose* a comenzii **gcc** prezintă detaliat fazele procesului de compilare:

```
1  razvan@valhalla:~/code/ptr$ gcc -v -Wall ptr.c -o ptr
2  Using built-in specs.
3  Target: x86_64-linux-gnu
4  [...]
5   /usr/lib/gcc/x86_64-linux-gnu/4.3.3/cc1 -quiet -v ptr.c -quiet -dumpbase
   ptr.c -mtune=generic -auxbase ptr -Wall -version -o /tmp/ccYyQvdx.s
6  [...]
7  GNU C (Debian 4.3.3-14) version 4.3.3 (x86_64-linux-gnu) compiled by GNU
   C version 4.3.3, GMP version 4.3.1, MPFR version 2.4.1-p2.
8  [...]
9   as -V -Qy -o /tmp/ccoFhtN3.o /tmp/ccYyQvdx.s
10 GNU assembler version 2.19.51 (x86_64-linux-gnu) using BFD version (GNU
   Binutils for Debian) 2.19.51.20090723
11 [...]
12  /usr/lib/gcc/x86_64-linux-gnu/4.3.3/collect2 --build-id --eh-frame-hdr -
m elf_x86_64 --hash-style=both -dynamic-linker /lib64/ld-linux-x86-64.so
.2 -o ptr /usr/lib/gcc/x86_64-linux-gnu/4.3.3/../../../../lib/crt1.o
[...] /tmp/ccoFhtN3.o -lgcc --as-needed -lgcc_s --no-as-needed -lc -lgcc
[...]
```

După cum se observă mai sus, există 3 etape rezultate în urma rulării comenzii **gcc**:

1. invocarea compilatorului (**cc1**) conduce la obținerea fișierului în limbaj de asamblare `/tmp/ccYyQvdx.s`;

2. invocarea asamblorului (**as**) conduce la obținerea fișierului obiect `/tmp/ccoFhtN3.o`;
3. invocarea linker-ului (**collect2**) conduce la obținerea fișierului executabil `ptr`;

În cazul GCC, preprocesorul este invocat intern de compilator (**cc1**) și nu rezultă în obținerea unui fișier temporar.

În continuare vor fi prezentate mai detaliat fazele procesului de compilare și fișierele intermediare obținute.

11.3.1 Preprocesarea

Etapa de preprocesare presupune înlocuirea/expandarea directivelor de preprocesare din fișierul sursă.

Directivele de preprocesare încep cu `#` (*diez*, *sharp* în engleză). Printre acestea se numără directiva `#include` și directiva `#define`. După preprocesare, directiva `#include` este înlocuită cu fișierul inclus, iar directiva `#define` definește un macro care va fi substituit cu valoarea sa oriunde apare în codul sursă.

În exemplul de mai jos se realizează preprocesarea fișierului `info.c`.

```
1  #ifndef INFO_H_
2  #define INFO_H_          1
3
4  #define MAX_ARRAY_LENGTH        32
5
6  struct array_struct {
7          int array[MAX_ARRAY_LENGTH];
8          int len;
9  };
10
11 #endif
```

Listing 11.9: info.h

```
1  #include "info.h"
2
3  int get_sum(struct array_struct *a)
4  {
5          int i;
6          int sum;
7
8          if (a->len >= MAX_ARRAY_LENGTH)
9                  return 0;
10
11         for (i = 0; i < a->len; i++)
12                 sum += a->array[i];
13
14         return sum;
15 }
```

Listing 11.10: info.c

Pentru oprirea rulării GCC după faza de preprocesare se folosește opțiunea `-E`:

```
mircea@cougar:~/carte-uso/cap-10$ gcc -E info.c
# 1 "info.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "info.c"
# 1 "info.h" 1


struct array_strunct {
 int array[32];
 int len;
};
# 2 "info.c" 2

int get_sum(struct array_struct *a)
{
 int i;
 int sum;

 if (a->len >= 32)
  return 0;

 for (i = 0; i < a->len; i++)
  sum += a->array[i];

 return sum;
}
```

Se observă înlocuirea directivei `#include` cu fișierul `info.h` și înlocuirea macrodefiniției `MAX_ARRAY_LENGTH` cu `32` în fișierul preprocesat.

Preprocesarea este singura fază a procesului de compilare care nu rezultă în mod implicit într-un nou fișier. În absența opțiunii `-o`, rezultatul preprocesării este scris la ieșirea standard.

Folosirea opțiunii `-o` sau a operatorului de redirectare permite precizarea fișierului de ieșire. Convențional, fișierul preprocesat are extensia `.i`:

```
mircea@cougar:~/carte-uso/cap-10$ gcc -E info.c -o info.i

mircea@cougar:~/carte-uso/cap-10$ gcc -E info.c > info.i
```

Preprocesarea este efectuată pe un sistem Linux cu ajutorul utilitarului GNU CPP (C preprocessor)[1].

11.3.2 Compilarea

> **Compilarea** este etapa în care din fișierul preprocesat se obține un fișier în limbaj de asamblare.

Compilarea efectivă înseamnă traducerea codului din limbaj de nivel înalt în limbaj de asamblare. Limbajul de asamblare descrie instrucțiunile specifice procesorului

[1] http://gcc.gnu.org/onlinedocs/cpp/

sistemului fizic. Pentru oprirea rulării GCC după faza de compilare se folosește opțiunea `-S`.

În exemplul de mai jos, se compilează fișierul `info.c` din secțiunea anterioară și se obține, implicit, fișierul `info.s`. Se poate preciza explicit fișierul de ieșire cu ajutorul opțiunii `-o`.

```
mircea@cougar:~/carte-uso/cap-10$ ls info.*
info.c  info.h

mircea@cougar:~/carte-uso/cap-10$ gcc -S info.c

mircea@cougar:~/carte-uso/cap-10$ ls info.*
info.c  info.h  info.s
```

```
        .file   "info.c"
        .text
.globl get_sum
        .type   get_sum, @function
get_sum:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $20, %esp
        movl    8(%ebp), %eax
        movl    128(%eax), %eax
        cmpl    $31, %eax
        jle     .L2
        movl    $0, -20(%ebp)
        jmp     .L4
[...]
```

Listing 11.11: info.s

Pe sistemele Linux și Unix, fișierul în limbaj de asamblare are, de obicei, extensia `.s`. Pe sistemele Windows, un fișier în limbaj de asamblare are extensia `.asm`.

Limbaje de asamblare

Limbajul de asamblare este un limbaj de programare de nivel scăzut. Instrucțiunile sale (denumite și mnemonici) sunt de fapt reprezentarea simbolică a instrucțiunilor puse la dispoziție de procesorul sistemului. Exemple de mnemonici sunt `mov`, `cmp`, `add`, `sub`, `push` etc. Instrucțiunea în cod mașină:

```
10110000 01100001 (0xb061)
```

este scrisă în limbaj de asamblare ca:

```
mov     al,     061h
```

Mnemonica `mov` înseamnă efectuarea unei copieri. Rezultatul este încărcarea valorii `0x61` în registrul `al` al procesorului. Există, în general, o asociere unu la unu între o instrucțiune în limbaj de asamblare și o instrucțiune în cod mașină.

Un program scris în limbaj de asamblare este neportabil: poate fi rulat numai pe arhitectura de calcul pentru care a fost scris. Dacă programul în limbaj de asamblare trebuie portat pe o altă arhitectură, trebuie rescris și folosite mnemonicile spcifice acelei arhitecturi.

Întrucât necesită cunoștințe despre structura procesorului, resursele acestuia, mecanismul de acces la memorie, programarea în limbaj de asamblare este dificilă. Se recomandă folosirea acestuia doar la nevoie:

- când un limbaj cum este C nu oferă instrucțiuni pentru efectuarea unei anumite operații
- când este nevoie de eficiență maximă care poate fi obținută numai prin scrierea de cod în limbaj de asamblare

Traducerea codului în limbaj de asamblare în cod obiect se realizează cu ajutorul unui asamblor în faza de asamblare descrisă în secțiunea următoare. Fiind vorba de o asociere unu la unu între mnemonici și instrucțiunile procesorului, este posibilă obținerea codului în limbaj de asamblare pornind de la codul obiect. Această operație poartă numele de dezasamblare și este prezentată în secțiunea 11.3.3.

Tipuri de limbaje de asamblare

Chiar pentru același procesor pot exista limbaje de asamblare diferite sau, mai corect spus, sintaxe de limbaj de asamblare diferite. Pe arhitectura x86, asamblorul de la Microsoft folosește sintaxa Intel, pe când asamblorul de Linux (GNU) folosește sintaxa AT&T[1]. Astfel, copierea valorii `0x100` în registrul `eax` se face, folosind cele două sintaxe, astfel:

```
1 movl   $0x100,  %eax    ; sintaxa AT&T
2 mov    eax,     100h    ; sintaxa Intel
```

La fel, stocarea valorii `100` la adresa indicată de `eax`:

```
1 movl $0x100,    (eax)   ; sintaxa AT&T
2 movl [eax],     100h    ; sintaxa Intel
```

11.3.3 Asamblarea

Etapa de asamblare este etapa de traducere a codului scris în limbaj de asamblare (fișier text conținând mnemonici specifice arhictecturii sistemului de calcul) în cod binar. Acest cod binar este cod mașină reprezentând codificarea binară a instrucțiunilor procesorului. Fișierul obținut poartă numele de fișier cod obiect sau modul obiect. Pe Unix un modul obiect are extensia `.o` iar pe Windows extensia `.obj`.

Pentru oprirea rulării **gcc** după faza de asamblare se folosește opțiunea `-c`. În mod implicit se înlocuiește extensia fișierului sursă cu `.o`. Fișierul sursă poate fi un fișier C sau un fișier în limbaj de asamblare:

```
1 mircea@cougar:~/carte-uso/cap-10$ gcc -c info.c
2
3 mircea@cougar:~/carte-uso/cap-10$ ls -l info.o
4 -rw-r--r-- 1 ubuntu ubuntu 764 Sep 19 00:29 info.o
5
6 mircea@cougar:~/carte-uso/cap-10$ gcc -c info.s
7
```

[1] http://en.wikipedia.org/wiki/X86_assembly_language

```
mircea@cougar:~/carte-uso/cap-10$ ls -l
info.o -rw-r--r-- 1 ubuntu ubuntu 744 Sep 19 00:29 info.o
```

Ca și până acum, se poate folosi opțiunea `-o` pentru a preciza explicit numele fișierului de ieșire:

```
mircea@cougar:~/carte-uso/cap-10$ gcc -c info.s -o modul.o

mircea@cougar:~/carte-uso/cap-10$ ls -l modul.o
-rw-r--r-- 1 ubuntu ubuntu 744 Sep 19 00:47 modul.o
```

În cadrul suitei de aplicații GNU, asamblorul folosit Gas (GNU Assembler)[1], iar executabilul asociat este `as`. Un fișier în limbaj de asamblare poate fi asamblat folosind `as`:

```
mircea@cougar:~/carte-uso/cap-10$ as info.s -o as_out.o

mircea@cougar:~/carte-uso/cap-10$ ls -l as_out.o
-rw-r--r-- 1 ubuntu ubuntu 744 Sep 19 00:48 as_out.o
```

Modul obiect

Un modul obiect conține cod mașină specific procesorului. Acest cod este obținut în urma fazelor de preprocesare, compilare și asamblare. Un modul obiect are, de obicei, extensia `.o`:

```
mircea@cougar:~/carte-uso/cap-10$ file info.o
info.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not
stripped
```

objdump; dezasamblare

Datorită asocierii unu la unu care există între codul obiect și codul în limbaj de asamblare, se poate realiza o operație de dezasamblare.

> **Dezasamblare** înseamnă "recuperarea" mnemonicilor în limbaj de asamblare pornind de la codul mașină; este operația inversă asamblării.

Utilitarul folosit în acest sens este **`objdump`**:

```
razvan@anaconda:~/uso/compilare$ objdump -d info.o
info.o:     file format elf32-i386

Disassembly of section .text:

00000000 <get_sum>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 ec 14                sub    $0x14,%esp
   6:   8b 45 08                mov    0x8(%ebp),%eax
   9:   8b 80 80 00 00 00       mov    0x80(%eax),%eax
   f:   83 f8 1f                cmp    $0x1f,%eax
```

[1] http://en.wikipedia.org/wiki/GNU_Assembler

```
  12:   7e 09                   jle    ld <get_sum+0x1d>
  14:   c7 45 ec 00 00 00 00    movl   $0x0,0xffffffec(%ebp)
[...]
```

nm

Un alt utilitar folositor în lucrul cu fișiere obiect este **nm**. Cu ajutorul acestuia se pot lista simbolurile dintr-un fișier obiect:

```
razvan@anaconda:~/uso/compilare$ nm info.o
00000000 T get_sum
```

nm poate oferi informații despre definirea sau nu a unui simbol într-un modul obiect:

```
razvan@anaconda:~/uso/compilare$ nm main.o
         U add
00000000 T main
         U printf
         U sub
```

Din rezultatul afișării se observă că simbolurile sunt marcate cu `U` sau `T`. `T` înseamnă că acel simbol este definit în zona de text (cod) a executabilului, iar `U` înseamnă că este nedefinit în modulul obiect primit ca argument. În exemplul nostru, modulul are definit simbolul `main` dar nedefinite simbolurile `add`, `sub` și `printf`. Aceste simboluri se găsesc în alte module obiect sau în biblioteci. Rezolvarea acestor simboluri se va realiza în momentul link-editării.

11.3.4 Optimizarea compilării

Un compilator translatează codul sursă în limbaj de asamblare, și după faza de asamblare, în cod mașină. Operațiile descrise într-un limbaj de nivel înalt pot fi traduse în diverse "scenarii" de instrucțiuni în limbaj de asamblare. Având cunoștințe intime despre arhitectura sistemului de calcul (setul de instrucțiuni, registrele procesorului și modurile de adresare), compilatorul poate alege un scenariu mai bun de translatare.

> Faza translatării în care compilatorul încearcă obținerea unui cod mașină mai bun poartă numele de **optimizarea codului**.

Denumirea de *optimizare* este ușor forțată. Un termen mult mai bun este cel de *îmbunătățire*; nu se poate spune că se obține *cod optim*. Datorită răspândirii acestei forme, atât în limba română cât și în limba engleză (*optimizing compiler*), se va folosi în continuare denumirea de *optimizare*.

Optimizarea are drept scop obținerea unui executabil care să ruleze cât mai rapid și/sau care să ocupe cât mai puțin spațiu.

Se poate întâmpla ca aceste două scopuri (viteză mare, spațiu ocupat mic) să fie în conflict; utilizatorul va putea preciza compilatorului care scop este mai important.

O dată cu dezvoltarea sistemelor embedded, un rol important pe care trebuie să-l îndeplinească executabilul obținut de compilator este consumul redus de putere.

Folosirea optimizării reprezintă un compromis între obținerea unui executabil mai rapid sau care ocupă mai puțin spațiu și un timp mai îndelungat de compilare. Din acest motiv, partea de dezvoltare a unei aplicații se realizează fără optimizare. Faza de optimizare este folosită în momentul în care aplicația este pregătită pentru lansare (release).

Pentru a justifica avantajele și dezavantajele folosirii optimizării, mai jos se prezintă două scenarii de compilare a unei aplicații C++:

- fără optimizare; se obține executabilul `bb-ssa-no-opt`;
- cu folosirea unor opțiuni de optimizare; se obține executabilul `bb-ssa-opt`.

Se poate observa, ca *dezavantaj* o creștere a timpului necesar pentru compilare (de la `4.1` secunde la `5.7` secunde), dar ca *avantaj* o scădere a dimensiunii executabilului (de la `328KB` la `140KB`):

```
1  razvan@valhalla:~/school/2001-2006_code/pt/tema4/pt4$ time g++ -o bb-ssa-
   no-opt bb-ssa.cpp basic_block.cpp ssa.cpp asamblor.c analizor.c
   interpretor.tab.c -lfl
2
3  real    0m4.143s
4  user    0m3.624s
5  sys     0m0.488s
6
7  razvan@valhalla:~/school/2001-2006_code/pt/tema4/pt4$ time g++ -O -o bb-
   ssa-opt bb-ssa.cpp basic_block.cpp ssa.cpp asamblor.c analizor.c
   interpretor.tab.c -lfl
8
9  real    0m5.769s
10 user    0m5.288s
11 sys     0m0.456s
12
13 razvan@valhalla:~/school/2001-2006_code/pt/tema4/pt4$ ls -lh bb-ssa-no-
   opt
14 -rwxr-xr-x 1 razvan razvan 328K Aug 21 22:11 bb-ssa-no-opt
15
16 razvan@valhalla:~/school/2001-2006_code/pt/tema4/pt4$ ls -lh bb-ssa-opt
17 -rwxr-xr-x 1 razvan razvan 140K Aug 21 22:12 bb-ssa-opt
```

GCC permite optimizarea codului obținut prin intermediul opțiunii `-O`. Aceasta este urmată de un număr care specifică gradul de optimizare/îmbunătățire a codului. În mod implicit, compilatorul nu optimizează codul. Acest lucru este echivalent cu folosirea opțiunii `-O0` (litera `o` majusculă și `zero`):

```
1  mircea@cougar:~/carte-uso/cap-10$ gcc -c chatserv.c -O0
2
3  mircea@cougar:~/carte-uso/cap-10$ ls -l
4  chatserv.o -rw-r--r-- 1 ubuntu ubuntu 3656 Sep 18 23:46 chatserv.o
```

GCC prezintă 3 niveluri numerice de compilare:

- opțiunea `-O1` înseamnă reducerea timpului de execuție și a codului obținut fără a afecta semnificativ timpul de compilare; este echivalentă cu folosirea opțiunii `-O` (fără un argument numeric)

```
1  mircea@cougar:~/carte-uso/cap-10$ gcc -c chatserv.c -O1
2
3  mircea@cougar:~/carte-uso/cap-10$ ls -l
4  chatserv.o -rw-r--r-- 1 ubuntu ubuntu 3200 Sep 18 23:46 chatserv.o
```

- opțiunea `-O2` înseamnă folosirea tuturor optimizărilor care nu atrag un compromis viteza-spațiu

```
mircea@cougar:~/carte-uso/cap-10$ gcc -c chatserv.c -O2

mircea@cougar:~/carte-uso/cap-10$ ls -l
chatserv.o -rw-r--r-- 1 ubuntu ubuntu 3232 Sep 18 23:47 chatserv.o
```

Se observă că s-a obtinut un cod ceva mai mare decât în cazul opțiunii `-O1`, dar care va rula mai rapid. Nu întotdeauna un cod mai mic va fi mai rapid.

- opțiunea `-O3` este în acest moment cea mai puternică (și agresivă) metodă de compilare:

```
mircea@cougar:~/carte-uso/cap-10$ gcc -c chatserv.c -O3

mircea@cougar:~/carte-uso/cap-10$ ls -l
chatserv.o -rw-r--r-- 1 ubuntu ubuntu 3308 Sep 18 23:47 chatserv.o
```

- GCC oferă și opțiunea `-Os` care înseamnă optimizarea codului pentru a ocupa cât mai puțin spațiu. O utilizare tipică este în cazul dispozitivelor embedded care dispun de resurse hardware reduse:

```
mircea@cougar:~/carte-uso/cap-10$ gcc -c chatserv.c -Os

mircea@cougar:~/carte-uso/cap-10$ ls -l
chatserv.o -rw-r--r-- 1 ubuntu ubuntu 2868 Sep 18 23:47 chatserv.o
```

11.3.5 Link-editarea

Un modul obiect este asociat unui singur fișier/modul sursă. Modulul obiect descrie datele și codul funcțiilor proprii. Varibialele și funcțiile poartă numele generic de simboluri. Simbolurile externe modulului (adică funcțiile sau variabilelele externe – nedefinite local, prezente în alte module) sunt marcate ca nedefinite în cadrul modulului.

Pentru obținerea unui fișier executabil și deci pentru rularea acestui cod este necesară:

- identificarea simbolurilor nedefinite, operație denumită și rezolvarea simbolurilor, și
- unificarea sau legarea (linking) a zonelor de date și cod asociate într-un fișier executabil.

Această operație se numește link-editare, linking sau legare.

Spre exemplu, în momentul în care se compilează un modul obiect care folosește funcția de bibliotecă `printf`, simbolul apare ca fiind nedefinit (caracterul `U` la **nm**):

```
razvan@anaconda:~/uso/compilare$ nm main.o
00000000 T main
         U printf
```

Funcția `printf` este definită în biblioteca standard C. În urma link-editării se obține un executabil care importă zona de cod asociată funcției `printf` din biblioteca standard C. Altă situație este folosirea într-un modul obiect a unei funcții definite într-un alt modul

obiect; un exemplu este folosirea funcției `sum` în fișierul `main.c` descris în secțiunea 11.2.2. Funcția `sum` este definită în `sum.c`.

> Aplicația care rezolvă simbolurile nedefinite și care realizează legarea modulelor obiect poartă numele de **linker**.

Rolul unui linker este, așadar, de a lega mai multe fișiere obiect sau biblioteci, de a rezolva simbolurile nedefinite și de a obține un executabil sau o bibliotecă.

Figura 11.3 prezintă rolul îndeplinit de un linker.

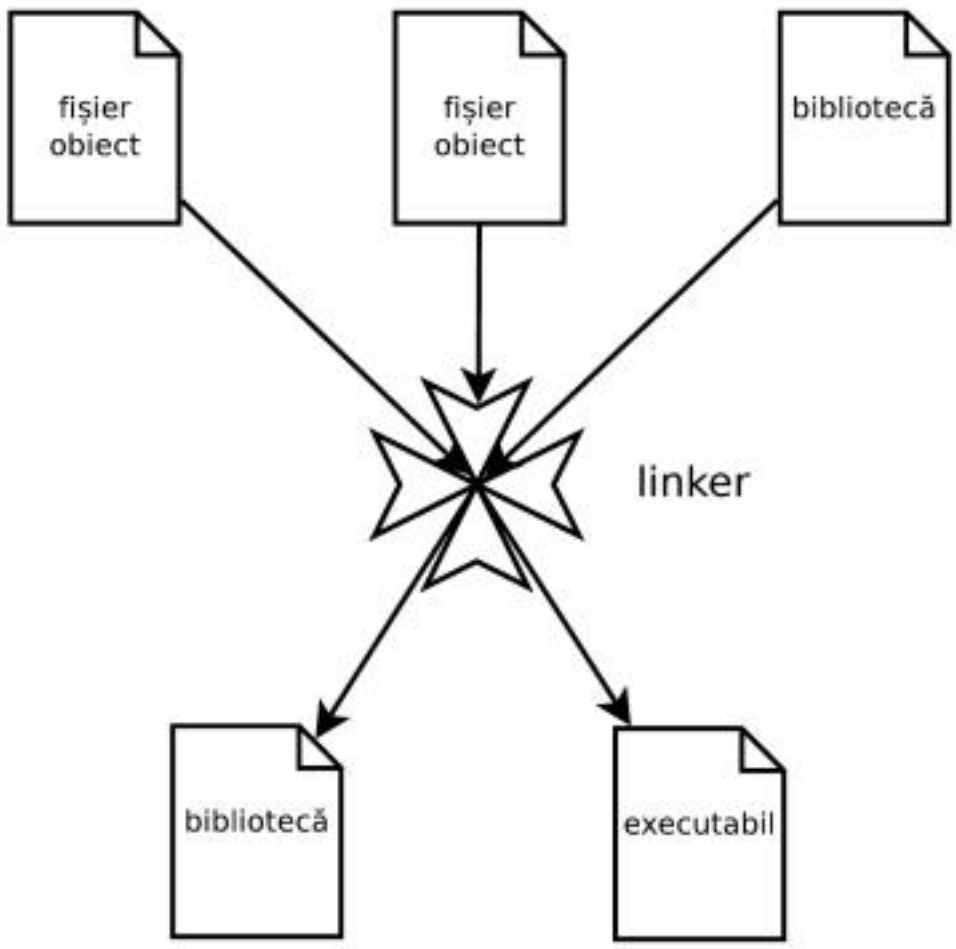


Figura 11.3: Funcționarea linker-ului

Mai multe detalii despre biblioteci se găsesc în secțiunea 11.4.

Invocare linker

Linker-ul de pe distribuțiile GNU/Linux este GNU linker sau GNU ld[1]. Executabilul asociat este **`ld`**. Linker-ul este rareori invocat cu ajutorul comenzii **`ld`**. În mod obișnuit, linker-ul este invocata intern de comanda **`gcc`**:

```
1  mircea@cougar:~/carte-uso/cap-10$ gcc -Wall -c main.c
2
3  mircea@cougar:~/carte-uso/cap-10$ gcc -Wall -c add.c
4
5  mircea@cougar:~/carte-uso/cap-10$ gcc -Wall -c sub.c
6
7  mircea@cougar:~/carte-uso/cap-10$ gcc main.o add.o sub.o -o exec
8
9  mircea@cougar:~/carte-uso/cap-10$ ./exec
10 Suma (10): 55
```

[1] http://en.wikipedia.org/wiki/GNU_linker

```
11 Suma alternanta (10): -5
```

Se poate folosi și **ld**, însă numărul mare de argumente care trebuie transmis face această opțiune mai puțin viabilă:

```
1 mircea@cougar:~/carte-uso/cap-10$ ld -dynamic-linker /lib/ld-linux.so.2 /
  usr/lib/crt1.o /usr/lib/crti.o /usr/lib/gcc/i486-linux-gnu/4.1.2/crtbegin
  .o main.o add.o sub.o -lc /usr/lib/gcc/i486-linux-gnu/4.1.2/crtend.o /usr
  /lib/crtn.o -o exec
2
3 mircea@cougar:~/carte-uso/cap-10$ ./exec
4 Suma (10): 55
5 Suma alternanta (10): -5
```

11.3.6 Fișiere executabile

> Un **fișier executabil** este un fișier binar obținut dintr-un set de fișiere obiect și biblioteci în urma operației de link-editare.

Spre deosebire de fișierele obiect, un fișier executabil are identificate și, în general, rezolvate simbolurile. Aceste simboluri sunt rezolvate la linking (legare statică) sau la rulare (legare dinamică).

În momentul lansării în execuție, informațiile conținute în fișierul executabil sunt folosite pentru a genera un proces. Un fișier executabil va conține codul mașină folosit pentru îndeplinirea sarcinilor date dar și antete și secțiuni de formatare auxiliare. Acestea sunt folosite pentru a descrie modul în care se folosește codul, zonele de memorie folosite etc.

Un fișier executabil are, așadar, un format bine definit. Formatul de fișier executabil se referă la fișiere obiect, fișiere executabile și la biblioteci cu legare dinamică.

Formate ale fișierelor executabile

Formatul unui fișier executabil[1] este strâns legat de sistemul de operare. Sistemul de operare este responsabil cu interpretarea unui fișier executabil și generarea unui proces pe baza acestuia. Un executabil poate fi rulat pe un sistem de operare dacă acel sistem de operare oferă suport pentru formatul de executabil folosit. Formatul de fișier executabil se referă și la fișierele obiect.

Exemple de formate de fișiere obiect/executabile sunt:

- **ELF** – folosit în sistemele Unix;
- **COFF** – versiunea anterioară ELF folosită pe Unix;
- **a.out** – primul format folosit de sistemele Unix; a oferit denumirea pentru executabilul implicit generat de compilator/linker;
- **PE** – formatul implicit pe sistemele Windows;

[1] http://en.wikipedia.org/wiki/Object_file

- **Mach-O** – formatul implicit pe Mac OS X.

În Linux, formatul de fișier obiect/executabil folosit este ELF (Executable and Linking Format)[1]:

```
1  mircea@cougar:~/carte-uso/cap-10$ file info.o
2  info.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not
   stripped
3
4  mircea@cougar:~/carte-uso/cap-10$ file exec
5  exec: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/
   Linux 2.4.1, dynamically linked (uses shared libs), for GNU/Linux 2.4.1,
   not stripped
6
7  mircea@cougar:~/carte-uso/cap-10$ file /bin/ls
8  /bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for
   GNU/Linux 2.4.1, dynamically linked (uses shared libs), for GNU/Linux
   2.4.1, stripped
9
10 mircea@cougar:~/carte-uso/cap-10$ file /lib/libc-2.3.6.so
11 /lib/libc-2.3.6.so: ELF 32-bit LSB shared object, Intel 80386, version 1
   (SYSV), for GNU/Linux 2.4.1, stripped
```

În listingul de mai sus se observă tipurile de fișiere care folosesc formatul ELF:

- module obiect; fișierul `info.o` este un modul obiect ELF relocabil[2];
- fișiere executabile; fișierele `exec` și `/bin/ls` sunt fișiere executabile format ELF;
- biblioteci partajate (vezi x.y.z); fișierul `/lib/libc-2.3.6.so` este o bibliotecă partajată (shared-object) (vezi secțiunea 11.4.1).

Utilitarul **`readelf`** poate fi folosit pentru analiza unui fișier în format ELF. În exemplul de mai jos se pot observa câteva din secțiunile unui fișier executabil: `.text` reprezintă zona de cod, `.rodata` zona de date read-only.

```
1  razvan@anaconda:~/uso/compilare$ readelf -S exec
2  There are 34 section headers, starting at offset 0xe38:
3
4  Section Headers:
5    [Nr] Name              Type            Addr     Off    Size   ES Flg Lk
    Inf Al
6    [ 0]                   NULL            00000000 000000 000000 00      0
     0  0
7    [ 1] .interp           PROGBITS        08048114 000114 000013 00   A  0
     0  1
8    [ 2] .note.ABI-tag     NOTE            08048128 000128 000020 00   A  0
     0  4
9    [ 3] .hash             HASH            08048148 000148 000028 04   A  4
     0  4
10 [...]
```

[1] http://en.wikipedia.org/wiki/Executable_and_Linkable_Format
[2] http://en.wikipedia.org/wiki/Relocation_(computer_science)

11.4 Biblioteci de funcții

O **bibliotecă** este colecție de funcții sau clase care oferă servicii preimplementate dezvoltatorului. În general, o bibliotecă este obținută prin comasarea mai multor fișiere obiect. O bibliotecă este, de asemenea, un fișier.

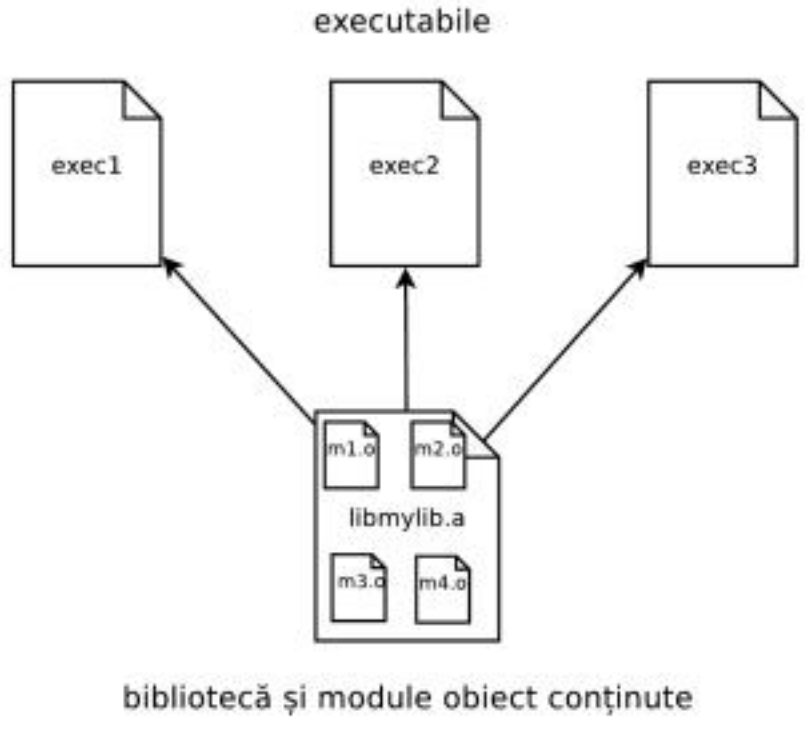


Figura 11.4: Bibliotecă

Bibliotecile de funcții (engleză: *libraries*) au aparut pentru a oferi o mai mare flexibilitate dezvoltatorilor de software. Acestea reunesc funcții des utilizate, astfel încât un program va folosi o bibliotecă fără a fi nevoie de rescrierea codului dorit.

În zilele noastre, foarte multe servicii au fost încapsulate în biblioteci de funcții pentru a ușura procesul de dezvoltare de noi aplicații. Astfel:

- biblioteca standard C (`libc`) oferă serviciile de bază ale sistemului;
- biblioteci precum GTK+[1] sau Qt[2] oferă primitive pentru dezvoltarea de interfețe grafice;
- utilitarul Pidgin[3] de instant messaging a izolat funcțiile principale în biblioteca `libpurple`;
- dezvotatorii Subversion[4] au dezvoltat biblioteca `libsvn1`;
- `libpng`, `libjpeg`, `libtiff` sunt folosite pentru prelucrarea de imagini;
- `libtorrent-rakshasa` și `libtorrent-rasterbar` sunt biblioteci care implementează protocolul BitTorrent folosite de un număr mare de clienți BitTorrent.

Mai sus sunt prezentate doar câteva dintre aplicațiile bibliotecilor. Practic, orice funcționalitate utilizabilă de mai multe aplicații poate fi implementată într-o bibliotecă.

[1] http://www.gtk.org/
[2] http://qt.nokia.com/
[3] http://www.pidgin.im/
[4] http://subversion.tigris.org/

Din circa 25000 de pachete instalabile în cadrul distribuției Debian GNU/Linux Testing/Squeeze[1], peste 5000 sunt biblioteci:

```
1 razvan@valhalla:~/code/ptr$ apt-cache search library | grep ^lib | wc -l
2 5699
```

11.4.1 Tipuri de biblioteci

Bibliotecile își încep utilizarea în momentul link-editării. Pentru rezolvarea simbolurilor nedefinite, linker-ul caută în modulele obiect încapsulate în cadrul bibliotecilor și realizează rezolvarea simbolurilor. Rezolvarea înseamnă marcarea locului în care simbolul este definit. În funcție de tipul bibliotecii, linker-ul decide să adauge modulul obiect necesar în codul executabilului sau doar să marcheze referința rezolvată. Există, astfel, două tipuri de biblioteci:

- **biblioteci cu legare statică** (*statically-linked libraries*) sau, pe scurt, *biblioteci statice*;
- **biblioteci cu legare dinamică** (*dinamically-linked libraries*) denumite *biblioteci partajate* (*shared libraries*) pe Linux și *dynamic-link libraries* (DLL) pe Windows.

Indiferent de tip, bibliotecile de funcții există sub formă de fișiere independente în sistem.

Bibliotecile statice (static libraries) sunt acele biblioteci ale căror module obiect componente sunt incluse în fișierul executabil în momentul link-editării.

Altfel spus, un executabil obținut în urma legării cu o bibliotecă statică deține tot codul necesar pentru a rula. Dimensiunea executabilului este mărită prin includerea fișierelor obiect necesare din cadrul bibiotecii.

În cazul **bibliotecilor partajate (shared libraries)**, operația de link-editare doar marchează referințele ca fiind rezolvate. Modulele obiect nu sunt incluse în codul executabilului obținut; vor fi adăugate în momentul lansării în execuție sau în momentul rulării.

Codul asociat unei biblioteci partajate nu ajunge niciodată în cadrul unui executabil. Biblioteca va fi încărcată în memorie la nevoie și va fi folosită în cadrul procesului obținut în urma execuției. În funcție de momentul în care biblioteca va fi încărcată în memorie, există două subtipuri de biblioteci cu legare dinamică:

- **biblioteci cu încărcare la execuție** (*load-time dynamically-linked library*) – biblioteca este adusă în memorie în momentul în care programul executat, dacă nu există deja acolo;
- **biblioteci cu încărcare la rulare** (*run-time dinamically-linked library*) – biblioteca este adusă în memorie la cerere, în momentul în care programul execută o instrucțiune care solicită acest lucru.

Bibliotecile partajate, după cum le spune și denumirea, reprezintă metoda cea mai utilizată de a pune la dispoziție funcții comune pentru mai multe aplicații. Dacă două

[1] http://packages.debian.org/testing/

sau mai multe aplicații folosesc aceeași bibliotecă, aceasta va trebui să fie încărcată o singură dată în memorie, economisind astfel spațiul ocupat. Biblioteca standard C, folosită de cea mai mare parte a aplicațiilor, este, de obicei, o bibiotecă partajată.

Tabela 11.1 reprezintă o comparație între bibliotecile statice și cele partajate:

Tabelul 11.1: Comparație între tipurile de biblioteci

Biblioteci statice	**Biblioteci partajate**
Colecție/arhivă de module obiect	Fișier format ELF
Executabil rezultat mai mare; se include cod	Executabil de dimensiune mică; nu se include cod
Executabilul poate fi mutat pe alt sistem	Executabilul are nevoie de prezența bibliotecii pentru a putea rula
Codul necesar este adăugat fiecărui executabil și apoi fiecărui proces	Codul bibliotecii este partajat între toate procesele care o folosesc
Timp de execuție mai rapid	Timp de execuție mai lent; se face rezolvarea simbolurilor în mod dinamic

11.4.2 Informații despre bibliotecile de funcții

Pe un sistem Linux, bibliotecile se găsesc în directoarele `/lib`, `/usr/lib` sau `/usr/local/lib`. Bibliotecile statice au extensia `.a`, iar cele partajate `.so` (de la shared object).

Informații primare despre bibliotecile de funcții se pot afla prin intermediul comenzii **`file`**:

```
1 mircea@cougar:~/carte-uso/cap-10$ file /usr/lib/libm.a
2 /usr/lib/libm.a: current ar archive
3
4 mircea@cougar:~/carte-uso/cap-10$ file /lib/libm.so.6
5 /lib/libm.so.6: symbolic link to 'libm-2.3.3.so'
6
7 mircea@cougar:~/carte-uso/cap-10$ file /lib/libm-2.3.3.so
8 /lib/libm-2.3.3.so: ELF 32-bit LSB shared
9 object, Intel 80386, version 1 (SYSV), not stripped
```

Bibliotecile statice sunt de fapt arhive de fișiere obiect. Ele sunt create prin intermediul comenzii **`ar`**. (pentru mai multe informații se poate consulta man **`ar`**).

O bibliotecă partajată este creată de linker. Pe un sistem Linux, o bibliotecă partajată se creează cu ajutorul comenzii **`gcc`** și opțiunea `-shared`[1]. O bibliotecă partajată este identificată printr-un număr de versiune. Pentru a facilita folosirea acesteia, se creează o legătură care nu include numărul versiunii.

Comanda **`nm`** poate fi utilizată pentru a vizualiza lista simbolurilor dintr-o bibliotecă (statică sau dinamică):

```
1 mircea@cougar:~/carte-uso/cap-10$ nm /lib/libm.so.6
2 00052b60 T fscanf
3 00043f20 T fprintf
```

[1] http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html

```
00052b90 T scanf
00052bd0 T sscanf
         U __libc_enable_secure@@GLIBC_PRIVATE
         U __libc_stack_end@@GLIBC_2.1
[...]
```

Simbolurile marcate cu `T` sunt simboluri din zona de cod (funcții) ce pot fi folosite extern de alte programe sau alte biblioteci. Simbolurile care sunt precedate de caracterul `U` sunt simboluri care trebuie rezolvate de linker la apelarea programului (se observă că ele nu au adresă în fișierul cu biblioteca). Acestea nu sunt definite local, ci în alte biblioteci (simbolul `__libc_enable_secure@@GLIBC_PRIVATE` este definit în biblioteca standard C – libc)

11.4.3 Utilizarea bibliotecilor

Bibliotecile reprezintă colecții de module preimplementate folosite pentru obținerea unui executabil, ceea ce înseamnă că, în momentul link-editării, linker-ului îi trebuie precizate bibliotecile necesare. Acest lucru se realizează cu ajutorul opțiunii `-l` (litera "L mic"). În listingul 11.4.3 se apelează funcții din biblioteca `ncurses`.

```c
#include <stdio.h>
#include <stdlib.h>
#include <ncurses.h>

int main(void)
{
        initscr();
        cbreak(); noecho();
        printf("Smash␣forehead␣on␣keyboard␣to␣continue␣...");
        fflush(stdout); fflush(stdin);
        getch();
        nocbreak(); echo();
        endwin();

        return 0;
}
```

Listing 11.12: Folosire funcții din biblioteca ncurses

În urma comenzii de compilare și rulare rezultă următoarea eroare:

```
razvan@valhalla:~/carte-uso/cap-10$ gcc -Wall -o getch getch.c
/tmp/ccavsseB.o: In function 'main':
getch.c:(.text+0x5): undefined reference to 'initscr'
getch.c:(.text+0xa): undefined reference to 'cbreak'
getch.c:(.text+0xf): undefined reference to 'noecho'
getch.c:(.text+0x3d): undefined reference to 'stdscr'
getch.c:(.text+0x42): undefined reference to 'wgetch'
getch.c:(.text+0x47): undefined reference to 'nocbreak'
getch.c:(.text+0x4c): undefined reference to 'echo'
getch.c:(.text+0x51): undefined reference to 'endwin'
collect2: ld returned 1 exit status
```

Linker-ul nu găsește definițiile funcțiilor `initscr`, `cbreak`, `noecho` etc. Pentru a rezolva această problemă link-erului îi trebuie specificat să folosească biblioteca `ncurses`. Aceasta este localizată în fișierul `libncurses.a` sau `libncurses.so`:

```
razvan@valhalla:~/carte-uso/cap-10$ ls /usr/lib/libncurses.*
/usr/lib/libncurses.a  /usr/lib/libncurses.so  /usr/lib/libncurses.so.5
```

Numele sub care va fi folosită biblioteca de linker se obține prim eliminararea extensiei (`.a` sau `.so`) și a prefixului `lib`. Pentru face legarea fișierului `getch.c` cu biblioteca `ncurses`, se folosește comanda:

```
razvan@valhalla:~/carte-uso/cap-10$ gcc -Wall -o getch \
> getch.c -lncurses

razvan@valhalla:~/carte-uso/cap-10$ ls -l getch
-rwxr-xr-x 1 razvan razvan 10734 Sep 21 23:25 getch
```

Se observă că, în acest caz, operația de linking se desfășoară cu sucess și se obține executabilul `getch`.

Bibliotecile sunt căutate în zone standard din sistemul de fișiere: `/lib`, `/usr/lib`, `/usr/local/lib`. În cazul în care fișierul asociat bibliotecii nu se găsește într-unul din directorele standard, se folosește opțiunea `-L`. Astfel, dacă se dorește legarea cu o biblioteca definită în fișierul `/home/traian/libs/libavatar.a`, se folosește o comandă de forma:

```
ubuntu@ubuntu:~$ gcc -Wall -o morph morph.c -lavatar -L/home/traian/libs
```


Când se specifică un director suplimentar în care se va realiza căutarea, numele acestuia se "lipește" de opțiunea `-L`; **nu** se lasă spații libere între opțiunea `-L` și numele directorului.

În cazul bibliotecilor partajate, există o etapă suplimentară de căutare a bibliotecii în momentul lansării în execuție a programului (la încărcare). Modul în care se specifică directoare suplimentare de căutare depășește sfera de cuprindere a acestui capitol. Pentru cei interesați, Internetul oferă multe tutoriale despre aceste noțiuni[1].

11.5 Automatizarea sarcinilor – make

Aplicațiile complexe dispun de un număr mare de fișiere sursă. Compilarea și link-editarea acestora poate deveni un proces anevoios și repetitiv. Se poate întâmplă ca doar câteva fișiere sursă să fie modificate și să nu fie nevoie de recompilarea întregii aplicații. Se dorește, așadar, un utilitar care să automatizeze sarcinile de compilare, linking și altele și să rezolve dependențele între fișierele sursă pentru a eficientiza procesul.

Make este un un astfel de utilitar: un program pentru automatizarea task-urilor. Make rezolvă problema execuției unor acțiuni în funcție de relațiile (dependețele) între ele. O acțiune este executată doar dacă acțiunile de care depinde au fost executate.

Relațiile între acțiuni sunt definite într-un fișier numit `Makefile`. Pe baza acestuia, Make determină automat secvența de pași care trebuie efectuată pentru respectarea dependențelor. Pentru a optimiza execuția acțiunilor, Make determină dacă o acțiune a fost efectuată și nu o mai execută. În același timp, Make detectează dacă pentru o

[1] http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html

acțiune s-a modificat o dependență și o execută din nou. Figura 11.5 descrie mdoul de funcționare a utilitarului Make.

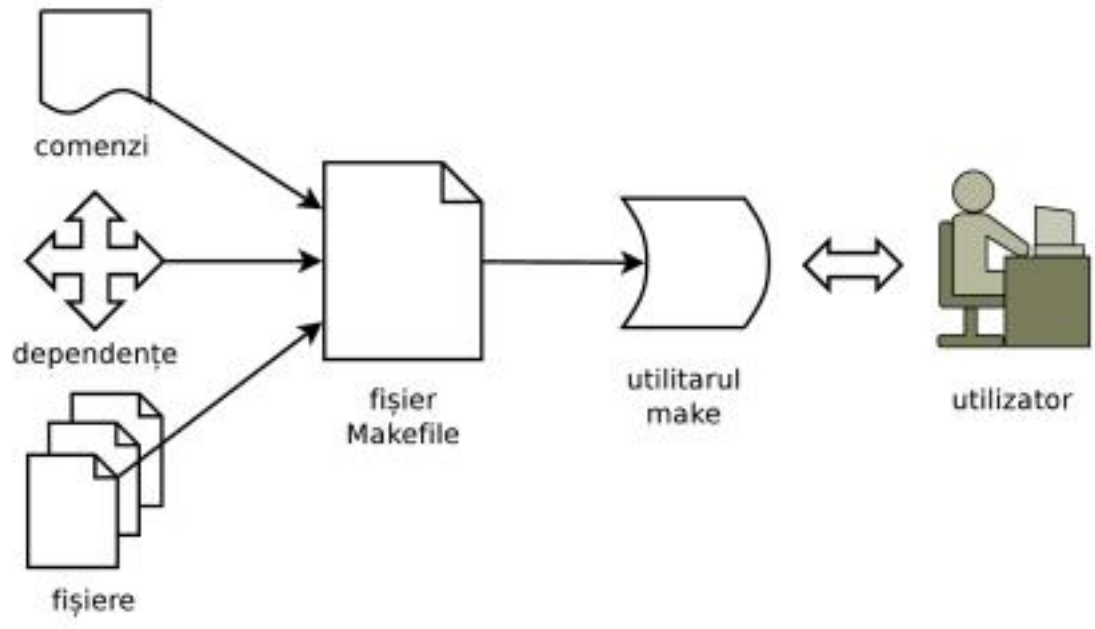


Figura 11.5: Funcționare Make

Make reprezintă, de fapt, o clasă de aplicații. Cel mai răspândit membru al acestei clase este GNU Make[1]. Executabilul asociat este `make`.

Vom descrie modul de funcționare și utilizare a Make prin compilarea și link-editarea aplicației descrisă în secțiunea 10.3. Aplicația constă din 4 fișiere (`main.c`, `sum.c`, `sumalt.c`, `func.h`).

11.5.1 Cel mai simplu Makefile

Cea mai simplă formă de fișier `Makefile` conține comanda de compilare și linking a fișierelor de mai sus:

```
1 exec:
2         gcc main.c sum.c sumalt.c -o exec
```

Listing 11.13: Makefile foarte simplu

Pentru rularea comenzii descrise în fișierul `Makefile` se folosește comanda **`make`**:

```
1 razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ make
2 gcc main.c sum.c sumalt.c -o exec
```

Fișierul Makefile specifică faptul că pentru a obține executabilul `exec` se rulează comanda **`gcc main.c sum.c sumalt.c -o exec`**. `exec` poartă numele de **target** (*țintă*) a fișierului Makefile: ceea ce se dorește obținut.

Din motive de tradiție, comenzile dintr-un fișier Makefile sunt precedate de caracterul `TAB`. O greșeală frecventă este folosirea de spații în loc de caracterul `TAB` înainte de folosirea unei comenzi. Apariția unui mesaj de forma celui de mai jos înseamnă, destul de probabil, omiterea folosirii caracterului `TAB`:

```
1 razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ make
2 Makefile:2: *** missing separator.  Stop.
```

[1] http://www.gnu.org/software/make/

11.5.2 Folosirea dependențelor

Fișierul Makefile de mai sus are un neajuns important: la orice rulare a comenzii Make se va executa comanda de compilare și linking indiferent dacă fișierele sursă au fost sau nu modificate. Acest lucru înseamnă realizarea unei acțiuni inutile.

Pentru a evita acest lucru, specificăm în Makefile faptul că executabilul `exec` se obține din cele trei fișiere sursă (`main.c`, `sum.c`, `sumalt.c`). Cele trei fișiere sursă sunt adăugate în lista de dependențe a executabilului (după caracterul `:` - *două puncte*). **Dependențele** sunt fișiere sau alte target-uri de a căror existență sau realizare depinde target-ul curent:

```
1 exec: main.c sum.c sumalt.c
2         gcc main.c sum.c sumalt.c -o exec
```

Listing 11.14: Makefile cu dependențe

În această situație, rularea comenzii **`make`** nu mai are ca efect compilarea și link-editarea celor trei fișiere sursă ci afișarea unui mesaj care menționează că executabilul `exec` există și nu au fost modificate fișierele de care depinde:

```
1 razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ make
2 make: 'exec' is up to date.
```

Dacă un fișier din lista de dependențe va fi modificat, Make va observa acest lucru și va rula din nou comanda. Un element target sau dependență a unui fișier Makefile are două stări:

- actualizat: fișierul asociat nu a fost modificat și dependențele sale sunt actualizate;
- neactualizat: fișierul asociat a fost modificat sau o parte din dependențele sale nu sunt actualizate.

11.5.3 Dependențe ierarhice

Fișierul Makefile anterior are încă neajunsuri. Un neajuns important este prezența fișierelor sursă în lista de dependențe.

Fie situația în care programatorul modifică fișierul `sum.c`. Rularea comenzii **`make`** va însemna următorul set de pași:

- se verifică dacă dependențele target-ului `exec` sunt actualizate;
- se observă că fișierul `sum.c` a fost modificat, deci dependența `sum.c` este neactualizată;
- target-ul `exec` este marcat ca neactualizat; trebuie rulată comanda asociată target-ului;
- se rulează comanda **`gcc main.c sum.c sumalt.c`**;
- comanda are drept consecință compilarea **tuturor** fișierelor din lista de dependențe și apoi link-editarea fișierelor obiect asociate.

Se efectuează, așadar, trei procese de compilare (pentru `main.c`, `sum.c`, `sumalt.c`) și un proces de link-editare a celor trei module obiect asociate. Totuși, întrucât doar fișierul `sum.c` a fost modificat, nu este nevoie de compilarea fișierelor `main.c` și `sumalt.c`.

Pentru a elimina acest neajuns, se modifică fișierul `Makefile` astfel:

```
1  exec: main.o sum.o sumalt.o
2          gcc main.o sum.o sumalt.o -o exec
3
4  main.o: main.c
5          gcc -c main.c -o main.o
6
7  sum.o: sum.c
8          gcc -c sum.c -o sum.o
9
10 sumalt.o: sumalt.c
11         gcc -c sumalt.c -o sumalt.o
```

Listing 11.15: Makefile cu dependențe ierarhice

În această situație, fișierul `Makefile` descrie o arborescență de dependențe, ca în figura 11.6.

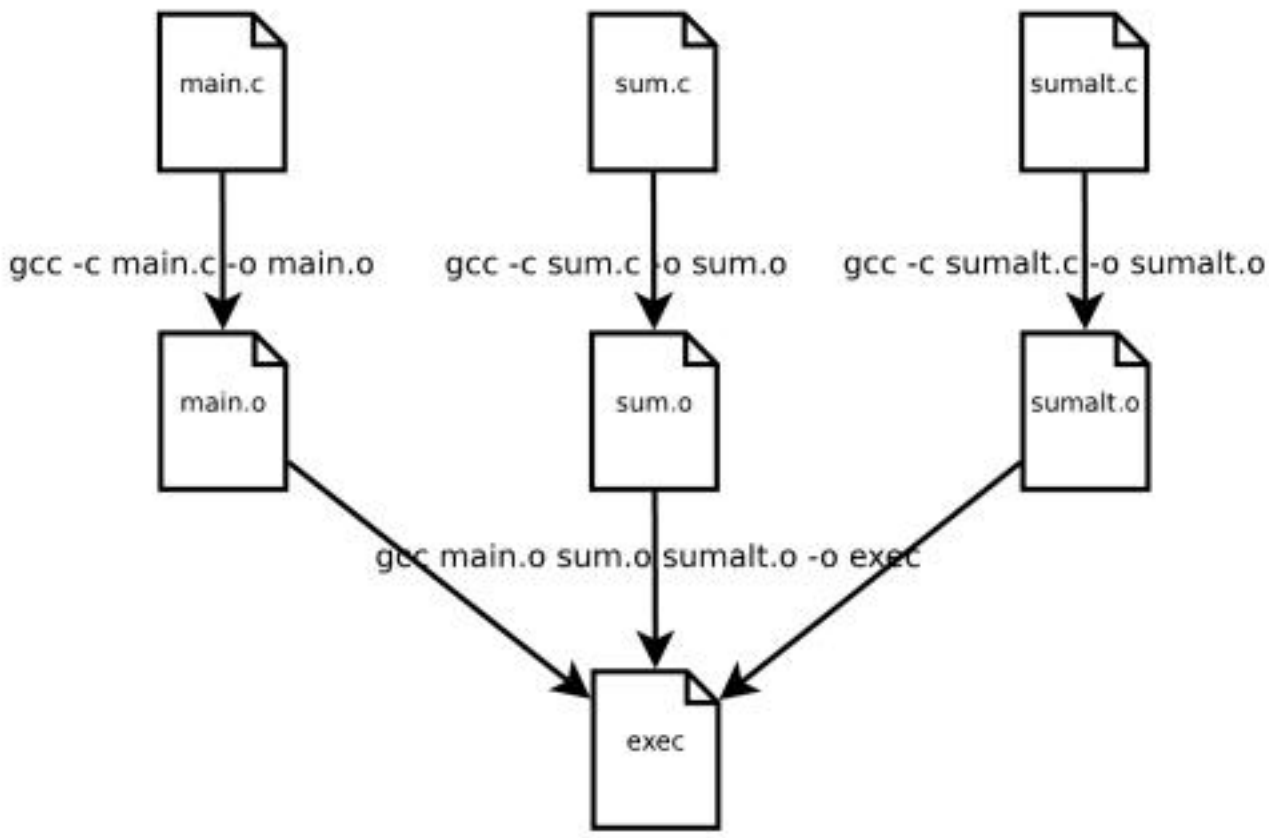


Figura 11.6: Dependențe într-un fișier Makefile

În figură sunt reprezentate și comenzile rulate pentru obținerea fiecărui fișier în parte. Astfel, o rulare simplă a comenzii **`make`** va produce următorul rezultat:

```
1  razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ make -f Makefile.
   dep.ier
2  gcc -c main.c -o main.o
3  gcc -c sum.c -o sum.o
4  gcc -c sumalt.c -o sumalt.o
5  gcc main.o sum.o sumalt.o -o exec
```

Pașii pe care îi efectuează utilitarul Make sunt:

- verifică lista de dependențe a target-ului `exec`; există trei dependențe care au la rândul lor alte dependențe; aceste dependențe trebuie verificate;
- target-ul `main.o` nu există, deci trebuie obținut; se rulează comanda **`gcc -c main.c -o main.o`**;
- target-ul `sum.o` nu există, deci trebuie obținut; se rulează comanda **`gcc -c sum.c -o sum.o`**;
- target-ul `sumalt.o` nu există, deci trebuie obținut; se rulează comanda **`gcc -c sumalt.c -o sumalt.o`**;
- toate cele trei dependențe ale target-ului `exec` erau neactuzalizate; target-ul `exec` este atunci neactualizat și trebuie rulată comanda asociată: **`gcc main.o sum.o sumalt.o -o exec`**.

După cum se precizase la început, presupunem că programatorul modifică fișierul `sum.c`. Rezultatul rulării comenzii **`make`** este:

```
1 razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ make -f Makefile.
  dep.ier
2 gcc -c sum.c -o sum.o
3 gcc main.o sum.o sumalt.o -o exec
```

Pașii urmați de utilitarul Make sunt:

- verifică lista de dependențe a target-ului `exec`; există trei dependențe care au la rândul lor alte dependențe; cele trei dependențe trebuie verificate;
- target-ul `main.o` există, dependența sa (`main.c`) există și este nemodificată; target-ul este actualizat, nu este nevoie de rularea comenzii asociate;
- target-ul `sum.o` există, dependența sa (`sum.c`) există **dar este modificată**; target-ul `sum.o` trebuie actualizat și se rulează comandă **`gcc -c sum.c -o sum.o`**;
- target-ul `sumalt.o` există, dependența sa (`sumalt.c`) există și este nemodificată; target-ul este actualizat, nu este nevoie de rularea comenzii asociate;
- una dintre trei dependențe ale target-ului `exec` (`sum.o`) este neactuzalizată; target-ul `exec` este atunci neactualizat și trebuie rulată comanda asociată: **`gcc main.o sum.o sumalt.o -o exec`**.

Se observă că, în acest caz, din cei patru pași posibili (trei pași de compilare și un pas de link-editare) este nevoie de execuția a doar doi dintre aceștia: compilarea `sum.c` și link-editarea.

Fișierul `Makefile` curent mai prezintă un neajuns: în cazul unei modificări a fișierul header `func.h` nu se actualizează niciun target, întrucât fișierul nu este adăugat în lista de dependențe a niciunui target. Un fișier `Makefile` corect arată astfel:

```
1 exec: main.o sum.o sumalt.o
2         gcc main.o sum.o sumalt.o -o exec
3
4 main.o: main.c func.h
5         gcc -c main.c -o main.o
6
```

```
7  sum.o: sum.c func.h
8          gcc -c sum.c -o sum.o
9
10 sumalt.o: sumalt.c func.h
11         gcc -c sumalt.c -o sumalt.o
```

Listing 11.16: Makefile cu fișier header în lista de dependențe

11.5.4 Target-ul clean

Pentru că, de obicei, se dorește curățarea completă a mediului de lucru (pentru distribuția surselor, mutarea aplicației, recompilarea acesteia în alt mediu etc.), un fișier Makefile oferă, de obicei, o regulă numită `clean`. Regula are rolul de a șterge fișierele produse în urma rulării comenzilor din fișierul Makefile: fișiere obiect, fișiere executabile, fișiere temporare etc.

În mod tipic, o regulă `clean` are asociată o comandă **`rm`** ca mai jos:

```
1  exec: main.o sum.o sumalt.o
2          gcc main.o sum.o sumalt.o -o exec
3
4  main.o: main.c
5          gcc -c main.c -o main.o
6
7  sum.o: sum.c
8          gcc -c sum.c -o sum.o
9
10 sumalt.o: sumalt.c
11         gcc -c sumalt.c -o sumalt.o
12
13 clean:
14         rm -f exec main.o sum.o sumalt.o
```

Listing 11.17: Makefile cu target clean

Pentru a rula comanda asociată target-ului `clean` se transmite target-ul ca argument comenzii **`make`**. Astfel, în urma rulării comenzii **`make clean`** se șterg fișierele obiect și fișierul executabil:

```
1  razvan@valhalla:~/carte-uso.git/src/code/10-compilare/tmp$ ls
2  Makefile  exec  func.h  main.c  main.o  sum.c  sum.o  sumalt.c  sumalt.o
3
4  razvan@valhalla:~/carte-uso.git/src/code/10-compilare/tmp$ make clean
5  rm -f exec main.o sum.o sumalt.o
6  razvan@valhalla:~/carte-uso.git/src/code/10-compilare/tmp$ ls
7  Makefile  func.h  main.c  sum.c  sumalt.c
```

11.5.5 Target-urile .PHONY și all

O problemă care poate împiedica funcționarea corectă a target-ului `clean` este prezența unui fișier cu numele `clean` în directorul curent. În această situația, Make va considera target-ul `clean` ca fiind actualizat și nu va rula comanda asociată.

```
1 razvan@valhalla:~/carte-uso.git/src/code/10-compilare/tmp$ ls
2 Makefile  clean  exec  func.h  main.c  main.o  sum.c  sum.o  sumalt.c
  sumalt.o
3
4 razvan@valhalla:~/carte-uso.git/src/code/10-compilare/tmp$ make clean
5 make: 'clean' is up to date
```

Pentru a rezolva această problemă, trebuie marcat `clean` ca un target care nu este niciodată actualizat. Comanda asociată va fi, astfel, executată tot timpul. Marcajul unui target ca *neactualizabil* se realizează cu ajutorul target-ului predefinit `.PHONY`, ca mai jos:

```
1 .PHONY: clean
2
3 clean:
4         rm -f exec main.o sum.o sumalt.o
```

Listing 11.18: Makefile cu target clean

Folosirea target-ului `.PHONY` înseamnă că se va rula comanda asociata target-ului `clean` indiferent dacă există sau nu un fișier cu numele `clean` în directorul curent:

```
1 razvan@valhalla:~/carte-uso.git/src/code/10-compilare/tmp$ ls
2 Makefile  clean  exec  func.h  main.c  main.o  sum.c  sum.o  sumalt.c
  sumalt.o
3
4 razvan@valhalla:~/carte-uso.git/src/code/10-compilare/tmp$ make clean
5 rm -f exec main.o sum.o sumalt.o
```

Target-ul `.PHONY` este, în general, folosit și pentru marcarea target-ului `all` ca target *neactulizabil*. Target-ul `all` este, în general, primul target într-un fișier Makefile și este verificat implicit la rularea comenzii **`make`**. Target-ul `all` încapsulează ca dependențe fișierele/target-urile ce se doresc obținute prin rularea comenzii **`make`** (în cazul de față este vorba doar de target-ul `exec`).

```
1  .PHONY: all clean
2
3  all: exec
4
5  exec: main.o sum.o sumalt.o
6          gcc main.o sum.o sumalt.o -o exec
7
8  main.o: main.c
9          gcc -c main.c -o main.o
10
11 sum.o: sum.c
12         gcc -c sum.c -o sum.o
13
14 sumalt.o: sumalt.c
15         gcc -c sumalt.c -o sumalt.o
16
17 clean:
18         rm -f exec main.o sum.o sumalt.o
```

Listing 11.19: Makefile cu target all

11.5.6 Variabile în Makefile

În fișierele Makefile construite până acum nu a fost folosită opțiunea `-Wall` la compilarea fișierelor sursă C. O soluție simplă este introducerea opțiunea `-Wall` în cadrul celor trei comenzi de compilare. Totuși, această abordare nu este flexibilă. Prespunând că, la un moment ulterior, se dorește și adăugare opțiunea `-g` pentru depanare, vor trebui din nou actualizate cele trei comenzi.

O abordare flexibilă este folosirea variabilelor în cadrul fișierului Makefile. O variabilă se definește o dată și se folosește de mai multe ori. Dacă, la un moment ulterior, se dorește actualizarea variabilei, modificarea se va face doar în locul unde a fost definită.

```
1  CC = gcc
2  CFLAGS = -Wall
3
4  exec: main.o sum.o sumalt.o
5          $(CC) main.o sum.o sumalt.o -o exec
6
7  main.o: main.c
8          $(CC) $(CFLAGS) -c main.c -o main.o
9
10 sum.o: sum.c
11         $(CC) $(CFLAGS) -c sum.c -o sum.o
12
13 sumalt.o: sumalt.c
14         $(CC) $(CFLAGS) -c sumalt.c -o sumalt.o
15
16 clean:
17         rm -f exec main.o sum.o sumalt.o
```

Listing 11.20: Makefile cu variabile

În cadrul fișierului Makefile de mai sus, au fost folosite variabilele, respectiv, `CC` pentru a reține comanda asociată compilatorului și `CFLAGS` pentru a reține opțiunile transmise acestuia. Variabila `CFLAGS` nu a fost folosită pentru obținerea executabilului `exec` pentru că are sens doar în faza de compilare, nu și în cea de link-editare (avertismentele indicate de `-Wall` rezultă în urma etapei de compilare).

11.5.7 Sintaxă Makefile

Această secțiune are un rol recapitulativ și descrie formal sintaxa unui fișier Makefile. După cum s-a prezentat și în secțiunile anterioare, sintaxa unui fișier Makefile este compusă dintr-o listă de reguli de forma:

```
1  target: dependente ...
2          <tab>comanda
3          <tab>comanda
4          <tab>...
```

Listing 11.21: Sintaxă Makefile

unde:

- **target** – numele unui fișier care trebuie generat sau numele unei acțiuni; în cazul în care target-ul este un fișier, comenzile trebuie să genereze acest fișier; în cazul

în care comenzile nu generează un fișier cu numele target-ului, se va considera la fiecare rulare ca target-ul nu a fost creat (așadar toate comenzile se vor executa);

- **dependențe** – lista de target-uri (separate prin spații) care trebuie îndeplinite (fișiere care trebuie să existe) pentru a se realiza target-ul curent – make poate determina dacă trebuie reexecutat target-ul dacă una dintre dependețe s-a modificat (unul din fișierele din lista de dependențe s-a modificat);
- **comandă** – comanda care duce la realizarea target-ului (de obicei la comenzi sunt trecute comenzile de compilare care duc la realizarea unui fișier cu numele target-ului);
- **<tab>** – caracterul `TAB`.

Regulile formează un arbore de dependențe. Pentru ca un target să fie executat, se verifică dacă toate dependențele lui există ca fișiere. Dacă nu este îndeplinită această condiție, dependențele care nu există sunt executate, aplicându-se același algoritm.

11.5.8 Moduri de utilizare a Make

La execuția comenzii **`make`**, vor fi luate în considerare target-urile definite în fișierul `Makefile`.

Pentru a executa un target, se folosește sintaxa:

```
1 mircea@cougar:~/carte-uso/cap-10$ make target
```

unde `target` este numele target-ului care se dorește a fi executat.

La utilizarea lui Make fără parametri, se va executa primul target definit în fișierul `Makefile`. Din acest motiv, primul target descris este de obicei cel mai complex, cu cele mai multe dependențe. Convențional, Makefile-urile folosite pentru compilarea unei aplicații definesc primul target `all`. Target-ul `all` deține în lista de dependențe celelalte target-uri care trebuie obținute.

Utilitarul GNU Make folosește, în mod implicit, unul din fișierele `GNUMakefile`, `Makefile` sau `makefile`. În cazul în care niciunul dintre acestea nu există, și comanda **`make`** nu primește nici un argument, va întoarce eroare:

```
1 razvan@valhalla:~/carte-uso.git/src/code$ make
2 make: *** No targets specified and no makefile found.  Stop.
```

Pentru a forța folosirea unui fișier Makefile cu alt nume decât numele implicite, se folosește opțiunea `-f`:

```
1 razvan@valhalla:~/carte-uso.git/src/code/10-compilare$ make -f Makefile.
  dep.ier.head
2 gcc -c main.c -o main.o
3 gcc -c sum.c -o sum.o
4 gcc -c sumalt.c -o sumalt.o
5 gcc main.o sum.o sumalt.o -o exec
```

Make poate funcționa și fără existența unui fișier Makefile. Pentru a compila direct fișierul `myprog.c` (care nu depinde de altele) se folosește comanda:

```
1 mircea@cougar:~/carte-uso/cap-10$ make myprog
```

Iar pentru a compila direct `myprog.c` cu parametrul de compilare `-Wall` se folosește comanda:

```
1 mircea@cougar:~/carte-uso/cap-10$ make CFLAGS=-Wall myprog
```

11.6 Portabilitate

Portabilitatea este caracteristica unei aplicații de a putea fi folosită într-un **mediu diferit** de cel pentru care fost inițial proiectată. Acest lucru poate însemna un alt sistem de operare, o altă arhitectură hardware, o altă bibliotecă.

Se spune că o aplicație este portabilă dacă are nevoie de modificări minime pentru a putea rula pe un alt mediu. Acțiunea de portare este acțiunea de modificare a unei aplicații pentru a putea fi folosită pe un alt sistem de operare sau sistem fizic. Un mediu pe care se realizează portarea se mai numește platformă.

Dacă în deceniile trecute numărul de arhitecturi posibile pe care rula o aplicație era destul de mare, în zilele noastre arhitectura hardware dominantă este x86. La fel sistemele de operare pe care se portează o aplicație sunt Microsoft Windows, Apple Mac OS X și Unix/Linux. În lumea sistemelor embedded, însă, portabilitatea rămâne o problemă importantă.

11.6.1 Portabilitatea la nivelul arhitecturii sistemului de calcul

În zilele noastre, sistemele desktop sunt dominate de arhitectura x86 așa că rareori este nevoie de portarea unei aplicații la nivelul arhitecturii sistemului de calcul. În sistemele embedded portarea la nivelul procesorului este un factor important.

În general, însă, portarea la nivelul arhitecturii nu este necesară datorită compilatorului. Astfel, pentru un limbaj de nivel înalt, compilatorul va asigura traducerea codului sursă în cod mașină specific procesorului dat. GCC este cel mai portat și cel mai portabil compilator fiind capabil de a compila programe pentru o varietate de arhitecturi.

11.6.2 Portabilitatea unui limbaj de programare

Programarea în limbaj de asamblare nu asigură, prin definiție, portabilitatea codului, deorece acesta este intrinsec legat de arhitectura sistemului de calcul. Unul dintre motivele răspândirii Unix în anii '70 a fost scrierea acestuia în C, un limbaj de nivel înalt portabil.

Folosirea C însemna eliberarea de arhitectura hardware pe care va rula aplicația. Portarea codului era făcută prin intermediul compilatorului. În ziua de azi, aplicațiile C sunt ușor portabile datorită existenței unui compilator C pe orice platformă. Cu toate acestea, anumite aspecte precum organizarea octeților[1] sau lungimea cuvântului

[1] http://en.wikipedia.org/wiki/Endianness

procesorului[1] rămân aspecte de care trebuie ținut cont.

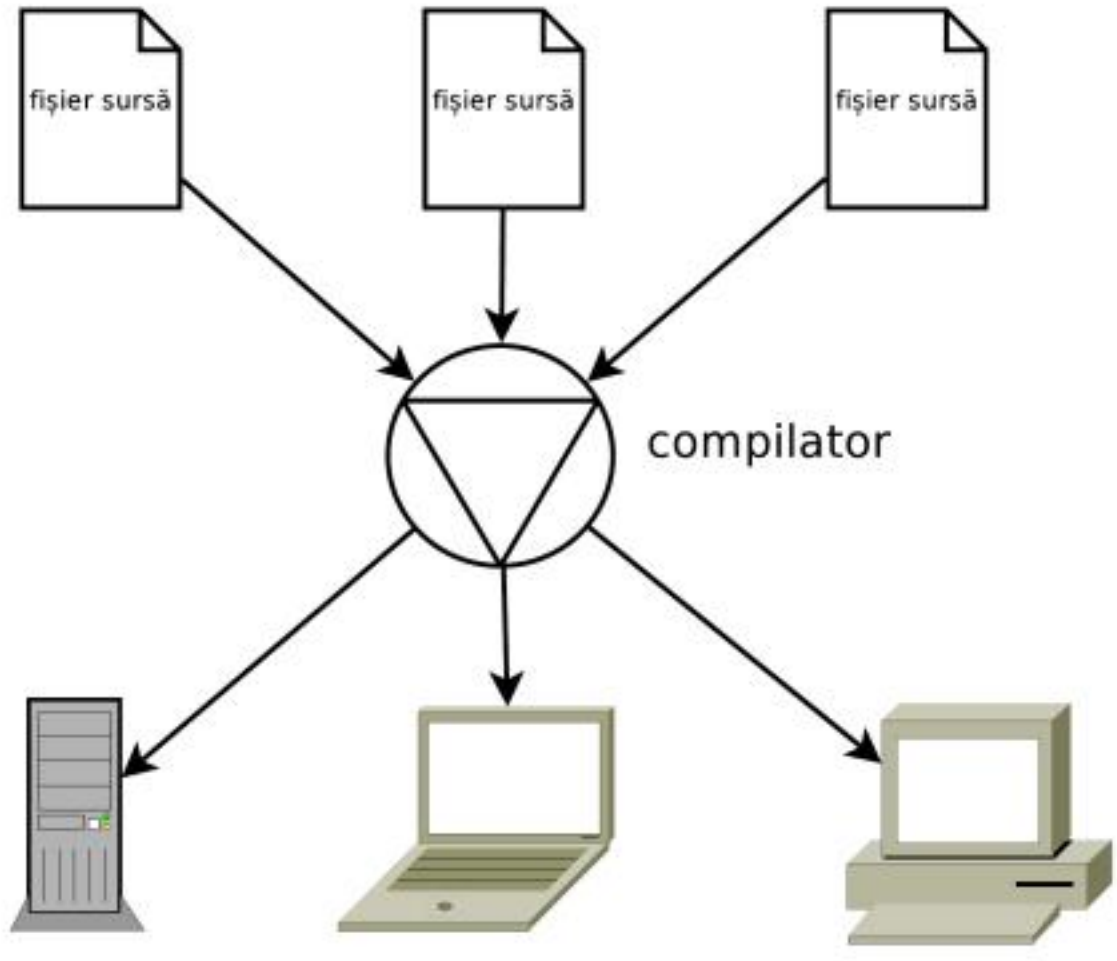


Figura 11.7: Portabilitatea asigurată de compilator

Java este un limbaj de programare gândit pentru a fi portabil. Deviza Java este "Write once, run everywhere". Pentru a asigura acest lucru, un program Java este compilat într-o formă portabilă de modul obiect denumită bytecode. Acest bytecode este apoi interpretat de o mașină virtuală Java. Mașina virtuală Java este responsabilă cu translatarea bytecode-ului portabil în cod specific sistemului pe care acesta rulează. Astfel, rularea unui program Java sau unui modul compilat în bytecode pe o nouă platformă este condiționată de existența unei mașini virtuale pe acea platformă. Mașina virtuală Java este conținută în pachetul JRE de la Sun (Java Runtime Environment). Acesta conține mașina virtuală și bibliotecile cu funcții standard Java.

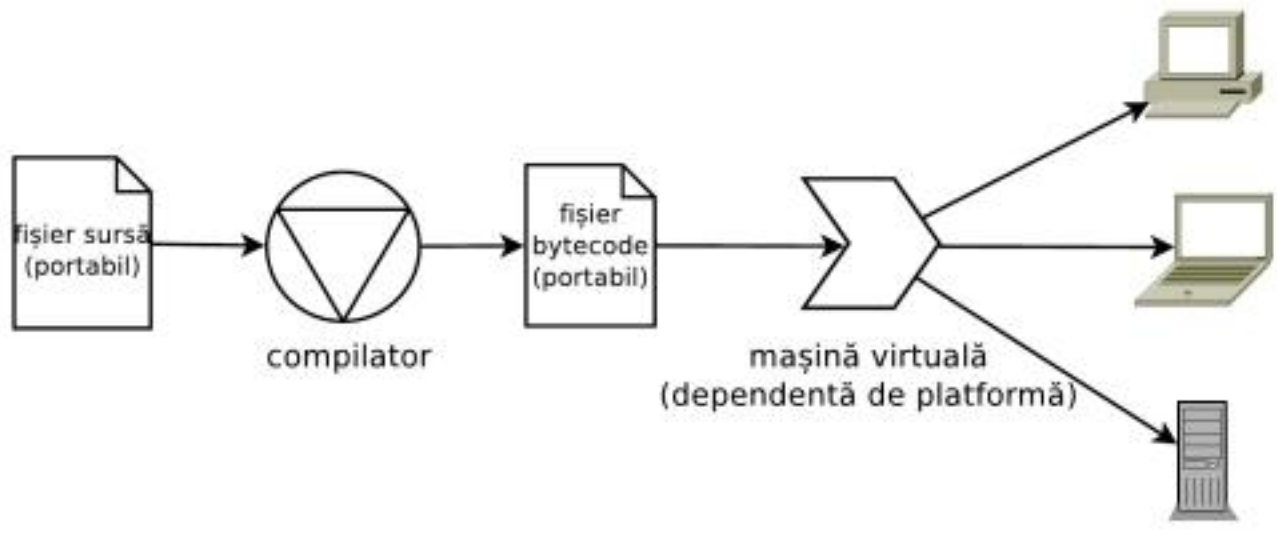


Figura 11.8: Portabilitatea asigurată de mașina virtuală

[1] http://en.wikipedia.org/wiki/Word_(computing)

11.6.3 Portabilitatea la nivelul sistemului de operare

Portabilitatea unei aplicații se poate referi și la posibilitatea rulării acesteia pe un alt sistem de operare. Acest lucru este corelat, de obicei, cu interfețele de programare (API – Application Programming Interface) pe care sistemul de operare le pune la dispoziție (system API). Astfel, Windows pune la dispoziția programatorului C interfața Win32 API[1]. Unix, pe de altă parte pune la dispoziția programatorului interfața POSIX (Portable Operating System Interface)[2]. Pentru deschiderea unui fișier un programator Windows va folosi apelul `CreateFile`, iar un programator Unix `open`.

Pentru asigurarea portabilității însă, multe biblioteci oferă programatorului o interfață portabilă peste sistemul de operare. Astfel, funcțiile ANSI din biblioteca standard C sunt portabile peste diverse sisteme de operare. De exemplu, pentru deschiderea unui fișier, programatorul va folosi apelul `fopen` indiferent de sistemul de operare pe care va rula aplicația.

Un exemplu de bibliotecă portabilă este wxWidgets[3]. Această bibliotecă permite crearea de aplicații grafice peste diferite sisteme de operare prin intermediul unui API comun. Biblioteca ascunde atât comunicarea cu sistemul de operare din spate cât și cu bibliotecile grafice neportabile pe care acesta le pune în mod obișnuit la dispoziție.

11.7 Studiu de caz

11.7.1 GCC în Windows

Suportul pentru compilatorul GCC în mediul Windows este disponibil prin proiectele:

- MinGW[4] (Minimalist GNU for Windows) reprezintă o portare a aplicațiilor de dezvoltare de bază ale proiectului GNU pe sisteme Windows. Componentele principale sunt GNU Compiler Collection (GCC), GNU Binutils și bibliotecile de runtime C. MinGW folosește DLL-urile de sistem standard și API-ul de bază al sistemului Windows.
- Cygwin[5] este un mediu care oferă aplicațiilor un strat (layer) de emulare a API-ului Unix, împreună cu o colecție de utilitare care oferă funcționalitățile din Unix. În vreme ce MinGW oferă doar aplicații pentru dezvoltare, Cygwin permite instalarea unei mari diversități de aplicații.

Aceste implementări oferă acces la suita de aplicații de dezvoltare a proiectului GNU. Parametrii și modul de utilizare a compilatorului sunt similare unui sistem Linux.

În mediul Windows este posibilă utilizarea GCC și împreună cu un mediu integrat de dezvoltare – IDE. Dev-C++[6] este un astfel de IDE care oferă posibilitatea compilării programelor utilizând GCC.

---

[1] http://msdn.microsoft.com/en-us/library/aa383749(VS.85).aspx
[2] http://www.opengroup.org/onlinepubs/9699919799/
[3] http://www.wxwidgets.org/
[4] http://www.mingw.org/
[5] http://sourceware.org/cygwin/
[6] http://www.bloodshed.net/devcpp.html

11.7.2 Link-editarea modulelor C și a modulelor C++

Deși două limbaje asemănătoare (C++ este considerat un superset al C), combinarea codului scris în C și C++ poate furniza probleme. În cele ce urmează nu ne vom referi la accesarea claselor C++ din C sau folosirea apelurilor din biblioteca standard C++ în programe C – situații rar întâlnite. Ne vom referi la folosirea funcțiilor definite în module C într-un modul C++ și invers[12].

Contextul

Vom considera un scenariu foarte simplu care include 6 fișiere:

- `c_print.c`: fișier C conține implementarea funcției `c_print`;
- `cpp_print.cpp`: fișier C++ conține implementarea funcției `cpp_print`;
- `c_print.h`: fișier header ce conține declarația funcției `c_print`;
- `cpp_print.h`: fișier header ce conține declarația funcției `cpp_print`;
- `c_main.c`: fișier C conținând funcția `main`; va apela funcția `c_print` din fișierul `c_print.c` sau funcția `cpp_print` din fișierul `cpp_print.cpp`;
- `cpp_main.cpp`: fișier C++ conținând funcția `main`; va apela funcția `c_print` din fișierul `c_print.c` sau funcția `cpp_print` din fișierul `cpp_print.cpp`.

```c
#include <stdio.h>

#include "c_print.h"

void c_print(void)
{
        printf("C-style: Wassup!\n");
}
```

Listing 11.22: c_print.c

```cpp
#include <iostream>

#include "cpp_print.h"

void cpp_print()
{
        std::cout << "C++ style: True, true!" << std::endl;
}
```

Listing 11.23: cpp_print.cpp

```c
#ifndef C_PRINT_H_
#define C_PRINT_H_      1

void c_print(void);

#endif
```

[1] http://en.wikipedia.org/wiki/Compatibility_of_C_and_C%2B%2B#Linking_C_and_C.2B.2B_code
[2] http://www.parashift.com/c++-faq-lite/mixing-c-and-cpp.html

Listing 11.24: c_print.h

```
#ifndef CPP_PRINT_H_
#define CPP_PRINT_H_    1

void cpp_print();

#endif
```

Listing 11.25: cpp_print.h

```
#include "c_print.h"
//#include "cpp_print.h"

int main(void)
{
        c_print();
        //cpp_print();

        return 0;
}
```

Listing 11.26: c_main.c

```
#include "cpp_print.h"
//#include "c_print.h"

int main()
{
        cpp_print();
        //c_print();

        return 0;
}
```

Listing 11.27: cpp_main.cpp

În mod implicit, în cadrul fișierului `c_main.c` se apelează funcția `c_print`, iar în cadrul fișierului `cpp_main.cpp` se apelează funcția `cpp_main`.

Într-un prim pas, se compilează cele patru fișiere C/C++. Folosim **gcc** pentru compilarea fișierelor C și **g++** pentru compilarea fișierelor C++.

```
razvan@valhalla:~/mix$ ls
c_main.c  c_print.c  c_print.h  cpp_main.cpp  cpp_print.cpp  cpp_print.h

razvan@valhalla:~/mix$ gcc -Wall -c c_main.c

razvan@valhalla:~/mix$ gcc -Wall -c c_print.c

razvan@valhalla:~/mix$ g++ -Wall -c cpp_main.cpp

razvan@valhalla:~/mix$ g++ -Wall -c cpp_print.cpp

razvan@valhalla:~/mix$ ls
c_main.c  c_print.c  c_print.o     cpp_main.o     cpp_print.h
c_main.o  c_print.h  cpp_main.cpp  cpp_print.cpp  cpp_print.o
```

Pentru link-editare, folosim perechile `c_main.o` și `c_print.o`, respectiv `cpp_main.o` și `cpp_print.o`. Executabilele obținute sunt denumite în conformitate cu fișierele obiect folosite pentru generare.

```
razvan@valhalla:~/mix$ gcc c_main.o c_print.o -o c_main-c_print

razvan@valhalla:~/mix$ g++ cpp_main.o cpp_print.o -o cpp_main-cpp_print

razvan@valhalla:~/mix$ ./c_main-c_print
C-style: Whassup?

razvan@valhalla:~/mix$ ./cpp_main-cpp_print
C++ style: True, true!
```

Pentru link-editarea modulelor obiect obținute din surse C++ a fost folosită comanda **g++**. Comanda **g++** apelează linker-ul (**ld**) cu toate opțiunile necesare. Se poate folosi și comanda **gcc** pentru linking, dar trebuie specificată explicit folosirea bibliotecii standard C++ (`libstdc++`).

```
razvan@valhalla:~/mix$ gcc cpp_main.o cpp_print.o -o cpp_main-cpp_print
cpp_main.o:(.eh_frame+0x12): undefined reference to `__gxx_personality_v0
'
cpp_print.o: In function `cpp_print()':
cpp_print.cpp:(.text+0xa): undefined reference to `std::cout'
[...]
collect2: ld returned 1 exit status

razvan@valhalla:~/mix$ gcc cpp_main.o
cpp_print.o -o cpp_main-cpp_print_new -lstdc++

razvan@valhalla:~/mix$ ./cpp_main-cpp_print_new
C++ style: True, true!
```

Problema

Ce se întâmplă, însă, dacă link-editarea se realizează între un modul obiect provenit dintr-un fișier sursă C și un modul provenit dintr-un fișier sursă C++?

Pentru a testa acest lucru, sunt actualizate fișierele `c_main.c`, respectiv `cpp_main.cpp` pentru a include și apela funcțiile specifice celuilalt limbaj. După care se compilează și se realizează link-editarea:

```
razvan@valhalla:~/mix-new$ gcc c_main.o cpp_print.o -o c_main-cpp_print -
lstdc++
c_main.o: In function `main':
c_main.c:(.text+0xa): undefined reference to `cpp_print'
collect2: ld returned 1 exit status

razvan@valhalla:~/mix-new$ g++ c_main.o cpp_print.o -o c_main-cpp_print -
lstdc++
c_main.o: In function `main':
c_main.c:(.text+0xa): undefined reference to `cpp_print'
collect2: ld returned 1 exit status

razvan@valhalla:~/mix-new$ g++ cpp_main.o c_print.o -o cpp_main-c_print
cpp_main.o: In function `main':
cpp_main.cpp:(.text+0x5): undefined reference to `c_print()'
```

```
14 collect2: ld returned 1 exit status
15
16 razvan@valhalla:~/mix-new$ gcc cpp_main.o c_print.o -o cpp_main-c_print -
   lstdc++
17 cpp_main.o: In function 'main':
18 cpp_main.cpp:(.text+0x5): undefined reference to 'c_print()'
19 collect2: ld returned 1 exit status
```

În ambele situații, indiferent de comanda folosită (**gcc** sau **g++**) se obține eroare de linker. Linker-ul nu găsește simbolul asociat funcției `cpp_print`, respectiv `c_print`.

Folosind utilitarul **nm** se pot inspecta simbolurile din cadrul celor patru module obiect:

```
1  razvan@valhalla:~/mix-new$ nm c_main.o
2                  U cpp_print
3  0000000000000000 T main
4
5  razvan@valhalla:~/mix-new$ nm cpp_main.o
6                  U _Z7c_printv
7                  U __gxx_personality_v0
8  0000000000000000 T main
9
10 razvan@valhalla:~/mix-new$ nm c_print.o
11 0000000000000000 T c_print
12                 U puts
13
14 razvan@valhalla:~/mix-new$ nm cpp_print.o
15 000000000000005f t _GLOBAL__I__Z9cpp_printv
16 0000000000000022 t _Z41__static_initialization_and_destruction_0ii
17 0000000000000000 T _Z9cpp_printv
18 [...]
```

Problema este folosirea tehnicii de *name mangling*[1] folosită de compilatorul de C++. Simbolurile dintr-un modul C++ sunt precedate de o construcție specială de formă `_Z`. Tehnica este folosită pentru a preveni folosirea aceluiași identificator pentru o variabilă funcție care are același nume în două namespace-uri diferite.

Modulul `c_main.o` marchează simbolul `cpp_print` nedefinit și solicită linker-ului rezolvarea acestuia. Modulul `cpp_print.o` definește însă simbolul `_Z9cpp_printv` și linker-ul nu poate face rezolvarea. Similar se întâmplă și în cazul simbolului `c_print` (varianta C) și `_Z7c_printv` (varianta C++).

Soluția

Pentru a rezolva această problemă, compilatorul trebuie să dețină informații suplimentare și să marcheze simbolurile corespunzătoare în modulele obiect.

O soluție directă este compilarea tuturor fișierelor sursă folosind **g++**. Compilatorul de C++ poate compila și fișiere C, iar modulele obținute vor folosi tehnica de *name mangling*.

```
1 razvan@valhalla:~/mix-new$ g++ cpp_main.cpp c_print.c -o cpp_main-c_print
2
3 razvan@valhalla:~/mix-new$ ./cpp_main-c_print
```

[1] http://en.wikipedia.org/wiki/Name_mangling

```
C-style: Whassup?

razvan@valhalla:~/mix-new$ g++ c_main.c cpp_print.cpp -o c_main-cpp_print

razvan@valhalla:~/mix-new$ ./c_main-cpp_print
C++ style: True, true!
```

Întrucât este posibilă absența unui anumit tip de compilator de pe un sistem și pentru că este mai eficient ca un fișier sursă într-un limbaj de prorgramare să fie compilat cu un compilator specific, este de dorit o altă soluție. Soluția constă în folosirea construcției `extern "C"`. Această construcție se plasează înaintea declarației unei funcții pentru a forța compilatorul de C++ să folosească un identificator de simbol care să nu utilizeze tehnica de *name mangling*.

În general, construcția `extern "C"` se folosește în cadrul fișierelor header unde se găsesc declarațiile de funcții. Fisierele `c_print.h` și `cpp_print.h` actualizate vor fi:

```
#ifndef C_PRINT_H_
#define C_PRINT_H_      1

#ifdef __cplusplus
extern "C" {
#endif

void c_print(void);

#ifdef __cplusplus
}
#endif

#endif
```

Listing 11.28: c_print.h cu extern "C"

```
#ifndef CPP_PRINT_H_
#define CPP_PRINT_H_    1

#ifdef __cplusplus
extern "C" {
#endif

void cpp_print();

#ifdef __cplusplus
}
#endif

#endif
```

Listing 11.29: cpp_print.h cu extern "C"

În acest moment se poate realiza link-editarea în "diagonală", adică folosind perechea `c_main.o` și `cpp_print.o` pe de o parte și perechea `cpp_main.o` și `c_print.o` de cealaltă parte.

```
razvan@valhalla:~/mix-new$ gcc -c c_main.c

razvan@valhalla:~/mix-new$ g++ -c cpp_main.cpp
```

```
4
5  razvan@valhalla:~/mix-new$ gcc -c c_print.c
6
7  razvan@valhalla:~/mix-new$ g++ -c cpp_print.cpp
8
9  razvan@valhalla:~/mix-new$ gcc c_main.o cpp_print.o -o c_main-cpp_print -
   lstdc++
10
11 razvan@valhalla:~/mix-new$ g++ cpp_main.o c_print.o -o cpp_main-c_print
12
13 razvan@valhalla:~/mix-new$ ./c_main-cpp_print
14 C++ style: True, true!
15
16 razvan@valhalla:~/mix-new$ ./cpp_main-c_print
17 C-style: Whassup?
```

Prin folosirea utilitarului **nm** se observă folosirea unui identificator de simbol fără *name mangling* în cadrul fișierelor obiect C++.

```
1  razvan@valhalla:~/mix-new$ nm cpp_main.o
2                   U __gxx_personality_v0
3                   U c_print
4  0000000000000000 T main
5
6  razvan@valhalla:~/mix-new$ nm cpp_print.o
7  000000000000005f t _GLOBAL__I_cpp_print
8  [...]
9  0000000000000000 T cpp_print
```

Construcția `extern "C"` se folosește atât în cazul în care se apelează funcții C dintr-un modul C++ cât și în cazul apelării unei funcții C++ dintr-un modul C.

În primul caz, cel al apelării funcției C dintr-un modul C++, compilatorul de C++ marchează în modulul obiect obținut din fișierul C++ simbolul ca fiind nedefinit, în forma non-mangled. În cel de-al doilea caz, al apelării unei funcții C++ dintr-un modul C, compilatorul de C++ marchează în modulul obiect obținut din fișierul C++ simbolul ca fiind definit, în formatul non-mangled.

Compilatorul ia această decizie pe baza folosirii construcției `extern "C"`. Este important ca atât fișierul care definește funcția, cât și cel care apelează funcția să includă fișierul header care conține declarația funcției precedată de construcția `extern "C"`.

Folosirea macro-ului `__cplusplus` permite diferențierea între compilatorul de C și compilatorul de C++. Doar compilatorul de C++ va folosi construcția `extern "C"`. Construcția nu are nicio relevanță pentru compilatorul de C.

Cuvinte cheie

- compilare
- executabil
- interpretare
- limbaj
- proces
- sursă
- preprocesare
- asamblare

- cod obiect
- linker
- bibliotecă
- partajare
- portabilitate
- execuție
- apel de sistem
- make
- target
- dependență

Întrebări

1. Bibliotecile partajate de funcții:
   - ❑ sunt încărcate explicit la cerere de aplicații
   - ❑ ajută ca aplicațiile să fie mai simplu de întreținut
   - ❑ măresc aplicațiile deoarece sunt incluse în executabilele rezultate
   - ❑ sunt prezente sub formă de arhive de fișiere obiect
2. Pentru a dezactiva link-editarea implicită pe care o realizează `GCC`, se utilizează parametrul:
   - ❑ `-O0`
   - ❑ `-c`
   - ❑ `-lstdc++`
   - ❑ `-o`
3. Caracteristica unei aplicații de a putea fi folosită într-un mediu pentru care nu a fost inițial proiectată:
   - ❑ se întâlnește doar la bibliotecile partajate
   - ❑ este întâlnită numai la aplicațiile Java
   - ❑ se numește portabilitate
   - ❑ niciuna din variantele de mai sus
4. Pentru a executa un target dintr-un fișier `Makefile`, se folosește comanda
   - ❑ **`make Makefile <target>`**
   - ❑ **`make <target>`**
   - ❑ **`<target>`**
   - ❑ **`make <target> Makefile`**
5. În urma preprocesării se obțin:
   - ❑ fișiere obiect
   - ❑ fișiere sursă C fără directive de precompilare
   - ❑ fișiere în limbaj de asamblare
   - ❑ fișiere executabile

6. Care din următoarele reprezintă o bibliotecă al cărei cod este integrat explicit în codul unui executabil?
   - ❑ static library
   - ❑ shared library
   - ❑ public library
   - ❑ common library
7. Apelul de sistem reprezintă o metodă de a accesa resursele computerului. Apelurile de sistem sunt realizate doar de biblioteci.
   - ❑ adevărat, adevărat
   - ❑ adevărat, fals
   - ❑ fals, adevărat
   - ❑ fals, fals
8. Fie comanda:

```
1 mircea@cougar:~/carte-uso/cap-10$ file /lib/libm-2.3.3.so
2 /lib/libm-2.3.3.so: ELF 32-bit LSB shared object, Intel 80386,
  version 1 (SYSV), not stripped
```

   Care afirmație referitoare la fișierul `/lib/libm-2.3.3.so` este falsă?
   - ❑ fișierul este binar, nu poate fi editat cu un editor text
   - ❑ fișierul este o bibliotecă partajată
   - ❑ fișierul este executabil
   - ❑ fișierul funcționează pe o platformă Intel pe 32 biți
9. Programele interpretate se depanează mai greu decât cele compilate. În urma compilării rezultă fișiere în limbaj de asamblare care sunt executate direct pe procesor.
   - ❑ adevărat, adevarat
   - ❑ adevărat, fals
   - ❑ fals, adevărat
   - ❑ fals, fals
10. Apelurile de sistem ale programului `<program>` pot fi urmărite folosind comanda:
    - ❑ **`time <program>`**
    - ❑ **`file <program>`**
    - ❑ **`ldd <program>`**
    - ❑ **`strace <program>`**

Capitolul 12

Shell scripting

Shell to DOS... Come in DOS, do you copy? Shell to DOS...

Ce se învață din acest capitol?

- Rolul unui shell în sistem
- Ce este un script shell
- Utilitatea scripturilor shell
- Interacțiunea cu shell-ul; execuția comenzilor într-un shell
- Crearea și rularea unui script shell
- Programarea shell: variabile, funcții, structuri de control
- Folosirea filtrelor de text
- Expandarea shell
- Programarea shell (batch programming) pe Windows
- Utilitarul sed
- Utilitarul awk

12.1 Noțiuni introductive

Capitolul de față își propune prezentarea mecanismelor prin care diversele comenzi utilizate până acum pot fi combinate pentru obținerea unor efecte complexe în cadrul unui script shell.

> Un **script shell** este un program scris într-un fișier text format din combinații de comenzi și instrucțiuni specifice unui interepretor de comenzi (unui shell).

Un **shell** este un program care asigură utilizatorului o interfață de control și utilizare a sistemului de operare.

Un shell poate avea interfață grafică (GUI) sau poate avea interfață text (CLI). În cea de-a doua situație un shell mai poartă denumire de interpretor de comenzi. Shell-uri cu interfață grafică sunt *Windows Explorer* sau medii desktop construite peste *X Window System*. Shell-uri cu interfață text (interpretoare de comenzi) sunt **`cmd.exe`** (Windows NT, XP), Windows PowerShell (Windows Vista, Server 2008, 7), *Bash* (Bourne Again Shell), *csh* (C Shell).

Noțiunea de shell scripting se aplică shell-urilor cu interfață text (*Bash*, *C shell* etc.). În capitolul de față, shell-ul folosit va fi *Bash*. *GNU Bash*[1] (Bourne-Again Shell) este shell-ul implicit pe majoritatea distribuțiilor de Linux și pe Mac OS X. Ne vom referi prin *Bash* (cu majusculă) la shell, iar cu **`bash`** la executabilul utilizat de acesta.

12.1.1 De ce shell scripting?

Utilitatea principală a unui shell script derivă din faptul că este o combinație de comenzi deja existente. Acest lucru înseamnă posibilitatea de automatizare. Sarcinile repetitive pot fi descrise într-un script shell. Ori de câte ori este necesară executarea acelei sarcini, se rulează script-ul shell și aceasta va fi îndeplinită.

Un exemplu elocvent este crearea unui cont de utilizator. Pașii urmați pentru aceasta pot fi următorii:

1. crearea unui cont de utilizator (folosind **`useradd`**);
2. copierea unor directoare și fișiere pentru contul proaspăt creat;
3. stabilirea de cote (*quota*);
4. schimbarea/stabilirea parolei;
5. transmiterea unui e-mail pentru a notifica utilizatorul de crearea contului.

În lipsa unui script shell, administratorul sistemului ar trebui să ruleze comenzi pentru fiecare din pașii de mai sus ori de câte ori este necesară crearea unui cont de utilizator. Folosind un script shell, administratorul îl va rula doar pe acesta (folosind eventuali parametri). Efectul imediat este mărirea eficienței.

Scripturile shell se folosesc în general pentru automatizarea sarcinilor de rutină, precum: realizarea de backup-uri, verificarea stării sistemului etc.

Există situații când un utilizator dorește realizarea unei noi comenzi prin îmbinarea unora deja existente. Pentru a realiza acest lucru, el va crea un script shell pe care-l va putea utiliza ulterior.

În sistemele Unix, scripturile shell au un grad mare de utilizabilitate pentru că cea mai mare partea a comenzilor urmează filozofia Unix: "*Do one thing, do one thing well*". Acest lucru însemnă că fiecare comandă este utilă și (am putea spune) excelează în

[1] http://www.gnu.org/software/bash/

realizarea unei anumite sarcini, urmând ca sarcinile mai complexe să fie realizate prin "îmbinarea" mai multor comenzi simple într-un script shell.

12.1.2 Facilități oferite de scripturile shell

În afara folosirii comenzilor simple deja existente în sistem, lucru care conduce la posibilitatea de automatizare a sarcinilor, un script shell oferă și alte facilități utilizatorului.

În primul rând, un shell oferă un limbaj de programare cu variabile și instrucțiuni de ciclu (`for`, `while`) și de decizie (`if`, `case`). Acest lucru înseamnă că, dincolo de comenzile de bază, un script shell poate folosi facilitățile tipice unui limbaj de programare.

Evident, se poate pune întrebarea: De ce nu s-ar folosi un limbaj de programare precum C/Java?

Avantajul major unui script shell este timpul redus de scriere a acestuia, utilizarea unor componente deja create (fără a fi nevoie de rescrierea acestora) și folosirea unor resurse specifice ale sistemului: fișiere de configurare, drepturi de acces, utilizatori etc. Folosind comenzi deja existente, facilități de combinare a acestora și de control al fluxului, un script shell este o soluție rapidă și eficientă a unei probleme. Se spune că un script shell este o metodă "*quick and dirty*" de rezolvare a unor probleme care ar necesita efort mai mare dacă s-ar folosi alte limbaje. În plus depanarea unui script shell este extrem de facilă. În momentul în care un script shell nu funcționează corect se poate determina și corecta foarte rapid linia din script care a cauzat eroarea.

Dezavantajul unui script shell este viteza scăzută de execuție și lipsa accesului la structuri eficiente de calcul. Astfel, nu este recomandată folosirea unui script shell într-o situație în care viteza de execuție este un factor important. La fel, anumite probleme nu se pretează la a fi rezolvate cu un script shell din cauza lipsei de facilități. Problemele de algoritmică, efectuarea de calcule complexe, interacțiunea cu hardware-ul se realizează prin folosirea de limbaje specializate.

Totodată, un script shell permite interacțiunea între comenzi (sau mai bine zis între procesele create prin rularea unei comenzi), prin operatori specifici: | (pipe), ;, ||, && etc.

Alte facilități includ:

- posibilitatea reținerii ieșirii unei comenzi într-o variabilă pentru folosirea ulterioară a acesteia;
- posibilitatea redirectării intrării/ieșirii unei comenzi dintr-un/într-un fișier; acest lucru duce la prelucrarea foarte ușoară a fișierelor relevante în sistem (de exemplu: `/etc/passwd`, `/etc/shadow`);
- folosirea de comenzi non-interactive și programarea scriptului pentru execuție la un moment ulterior.

Se observă, așadar, că un script shell este un instrument foarte puternic pentru rezolvarea rapidă a diverselor probleme des întâlnite de utilizator. Programarea shell devine un factor important pentru o mai buna utilizare și gestiune a sistemului.

12.2 Interacțiunea cu shell-ul

Shell-urile cu interfață text interacționează cu utilizatorul prin intermediul comenzilor. Un shell oferă un prompt utilizatorului (în mod tipic acesta se termină cu un caracter de forma $, % sau #) unde se pot introduce comenzi. Execuția unei comenzi înseamnă, de obicei, generarea unui proces dintr-un executabil asociat comenzii.

O deprindere utilă în lucrul cu comenzile este prelucrarea șirului de caractere care le formează. Astfel, un utilizator avansat va edita comenzile sale și va folosi istoricul pus la dispoziție de un shell pentru a fi cât mai eficient. Shell-ul pune la dispoziția utilizatorului mai multe facilități de editare și utilizare rapidă a comenzilor.

12.2.1 Editarea comenzilor

O comandă este introdusă la promptul shell-ului prin folosirea tastelor corespunzătoare, după care se apasă ENTER și comanda va fi executată.

Există, însă, situații în care se greșește sau se uită un argument sau o literă și este nevoie de editarea acelei comenzi pentru obținerea efectului dorit. Editarea comenzii poate însemna adăugarea sau șteregerea unui caracter, a unui set de caractere, a unui argument al comenzii. În mod evident, la fel ca în cazul unui editor, este nevoie și de parcurgerea comenzii scrise.

Editarea comenzilor în Bash este realizată prin intermediul bibliotecii *readline*. Detalii despre aceasta găsiți la online[1]. Biblioteca *readline* este cea care asigură editarea unei comenzi, oferind interfața de control a terminalului.

Comenzile puse la dispoziție de readline pentru editarea comenzilor shell sunt inspirate din editorul **Emacs**. Urmând convenția **Emacs**, vom prescurta cu `C` tasta `Control` iar cu `M` tasta `Alt` (`Meta`). Prezentăm, în continuare, câteva dintre cele mai utile comenzi:

- `C-f`, `C-b` – un caracter înainte, respectiv înapoi (echivalent cu `Săgeată-Dreapta` și `Săgeată-Stânga`)
- `C-a`, `C-e` – început de linie, respectiv sfârșit de linie (echivalent cu `Home` și `End`)
- `M-f`, `M-b` – cuvânt înainte, respectiv înapoi; pe unele terminale (spre exemplu `gnome-terminal`) trebuie dezactivate scurtăturile terminalului pentru a permite comenzile *readline*
- `C-d` – șterge caracterul de sub cursor (echivalent cu `Delete`)
- `Backspace` – șterge caracterul de dinainte de cursor
- `M-d` – șterge până la sfârșitul cuvântului
- `M-Backspace` – șterge până la începutul cuvântului
- `C-k` – șterge până la sfârșitul liniei
- `C-w` – șterge până la începutul liniei
- `C-y` – lipește (*paste*, *yank*) ceea ce s-a șters ultima oară

[1] http://www.gnu.org/software/bash/manual/html_node/Readline-Interaction.html

- `C-_` (`Control-underscore`) – anulează (*undo*) ultima comandă de editare.

Pentru cei ce doresc acest lucru, editarea comenzilor folosind *readline* poate fi personalizată folosind un fișier de configurare. Pentru mai multe detalii consultați manualul[1].

12.2.2 Folosirea istoricului. Completare automată

Biblioteca *readline* memorează comenzile folosite în shell într-un istoric care poate fi folosit ulterior. Se pot folosi comenzi *readline* pentru parcurgerea comenzilor shell. Astfel `C-p` sau `C-n` parcurg comenzile anterioare sau ulterioare comenzii curente. Sunt echivalente cu `Săgeată-Sus` și `Săgeată-Jos`.

Istoricul este salvat la părăsirea shell-ului într-un fișier. În cazul **`bash`** acesta este `~/.bash_history`. La o nouă autentificare, shell-ul va încărca respectivele comenzi din history și utilizatorul le va putea reutiliza.

Căutarea comenzilor (*reverse search*)

În afara parcurgerii comenzilor din istoric, o opțiune foarte utilă este căutarea acestora. Pentru căutarea comenzilor în istoric se poate folosi comanda `C-r` (*reverse search*):

```
1 razvan@anaconda:~$ cd
2 (reverse-i-search)'chm': chmod o+w uploads/
```

În exemplul de mai sus, apăsarea `C-r` duce la schimbarea promptului. Utilizatorul poate introduce un șir de căutare pentru identificarea unei comenzi. Pentru fiecare caracter introdus, shell-ul va afișa prima comandă anterioră cea mai potrivită. Completarea șirului va duce la refacerea căutării. În afara introducerii șirului de căutare există câteva comenzi utile în modul de căutare:

- dacă se apasă (din nou) `C-r` se caută următoarea potrivire;
- dacă se apasă `ESC` sau `C-j` se încheie căutarea; comanda afișată la prompt va fi cea găsită;
- dacă se apasă `ENTER` se încheie căutarea și se execută comandă găsită;
- dacă se apasă `C-g` se întrerupe căutarea cu revenire la comanda de dinainte de căutare.

Completare automată (*autocompletion*)

O facilitate extrem de utilă în eficientizarea lucrului cu comenzile shell este cea de completare automată (*autocompletion*). Această facilitate înseamnă introducerea unui șir parțial de început al unei comenzi sau al unei intrări în sistemul de fișiere și apoi apăsarea tastei `TAB` pentru completarea automată. Spre exemplu, dacă cineva dorește să verifice server-ul de nume (DNS) al unui sistem, va folosi comanda:

[1] http://www.gnu.org/software/bash/manual/html_node/Readline-Init-File.html

```
ubuntu@ubuntu:~$ cat /etc/resolv.conf
```

Un calcul simplu arată că este nevoie de apăsarea a 21 de taste (cu ENTER) pentru execuția comenzii.

Putem folosi, în schimb, completarea automată și vom apăsa tastele în mod prezentat mai jos:

```
ubuntu@ubuntu:~$ cat /e<TAB>res<TAB>.<TAB><ENTER>
```

Un calcul la fel de simplu indică apăsarea a doar 14 taste (plus că tasta `TAB` devine foarte importantă și este apăsată mult mai ușor). Comanda afișată la prompt va fi identică cu cea de mai sus și, drept urmare, efectul va fi același.

În consecință, se recomandă insistent folosirea facilității de completarea automată. Această facilitate nu este prezentă numai la shell-urile unui sistem de operare ci și pentru comanda dispozitivelor de rețea și a altor echipamente care dispun de o interfață în linia de comandă. În plus, folosirea tastei `TAB` are un rol important în evitarea erorilor de scriere în cazul intrărilor sistemului de fișiere. Experiența personală a autorilor acestei cărți a surprins, nu de puține ori, crearea de fișiere cu nume "apropiate", dar nu identice, de cele reale (precum `/etc/rezolv.conf`, `/etc/resolv.comf`, `/etc/resolv.cf` etc.). Aceste erori ar fi putut fi evitate (dincolo de o mai mare viteză de utilizare a comenzilor) prin folosirea facilității de completare automată.

În situația în care există mai multe posibilități de completare, apăsarea tastei `TAB` nu are niciun efect. Totuși, se pot vizualiza toate aceste combinații prin apăsarea de două ori a tastei `TAB`. Astfel, apăsarea `z<TAB><TAB>` va duce la afișarea tuturor comenzilor care încep cu litera z:

```
razvan@anaconda:~$ z<TAB><TAB>
zcat         zegrep       zforce       zipgrep      zless
zcmp         zeisstopnm   zgrep        zipinfo      zmore
zdiff        zenity       zip          zipnote      znew
zdump        zfgrep       zipcloak     zipsplit     zsoelim
```

La fel, folosirea comenzii **`cat /<TAB><TAB>`** duce la afișarea tuturor intrărilor care se găsesc în directorul rădăcină al sistemului de fișiere:

```
razvan@anaconda:~$ cat /
.viminfo         home/            mnt/             tmp/
aquota.user      initrd/          opt/             usr/
bin/             initrd.img       proc/            var/
boot/            initrd.img.old   root/            vmlinuz
cdrom/           lib/             sbin/            vmlinuz.old
dev/             lost+found/      srv/
etc/             media/           sys/
```

Extinderea completării automate După cum s-a observat, completarea automată se referă la comenzi și la intrări în sistemul de fișiere. Totuși, aceasta poate fi extinsă prin intermediul unor scripturi personalizate la argumentele posibile ale unei comenzi.

Astfel fișierul `/etc/bash_completion` este folosit pentru extinderea completării automate. Pentru folosirea acestui script este necesară rularea comenzii:

```
ubuntu@ubuntu:~$ . /etc/bash_completion
```

Câteva linii din acest fișier:

```
1 complete -f -X '!*.@(sxi|sti|pps|ppt|pot|odp|otp)' ooimpress
```

Linia de mai sus înseamnă că dacă un utilizator folosește în linia de comandă **ooimpress**, argumentele posibile ale acesteia vor fi considerate doar fișierele cu extensiile precizate.

```
1 # user commands see only users
2 complete -u su usermod userdel passwd chage write chfn groups slay w
```

Linia de mai sus permite ca argumente numai nume de utilizatori pentru comenzile de lucru cu aceștia.

În afara extinderilor existente în `/etc/bash_completion`, utilizatorul își poate defini noi extinderi pentru a le integra în shell. Se recomandă ca acestea să fie plasate în directorul `/etc/bash_completion.d`. Generarea de noi extensii de completare depășește domeniul de interes al acestei cărți. Recomandăm celor interesați pagina web asociată proiectului [1] sau articolul legat de completarea automata de pe *Debian Administration*[2].

12.2.3 Comenzi interne (*built-in*) și comenzi externe

Comenzile care pot fi rulate dintr-un shell sunt de două tipuri:

1. **comenzi externe**: sunt rulate prin intermediul unui executabil
2. **comenzi interne**: sunt executate direct de shell fără invocarea unui alt program.

Comenzile interne (sau *built-in*) sunt utilizate pentru implementarea de facilități care sunt fie imposibil fie neconvenabil de realizat folosind programe separate. Un bun exemplu este comanda **cd** de schimbare a directorului curent. Această comandă nu poate fi folosită dintr-un executabil extern, întrucât la sfârșitul rulării comenzii s-ar pierde orice efect ar avea (și nu ar fi posibilă, astfel, schimbarea persistentă a directorului curent).

Exemple de comenzi externe și executabilul folosit pentru rularea acestora sunt:

- **ls** cu `/bin/ls`
- **file** cu `/usr/bin/file`
- **su** cu `/bin/su`
- **ifconfig** cu `/sbin/ifconfig`

Exemple de comenzi interne sunt:

- `:` – comanda nu face nimic, doar expandează variabile și realizează redirectările; astfel, rularea comenzii:

```
1 ubuntu@ubuntu:~$ : > out.txt
```

duce la trunchierea fișierului out.txt. Versiunea și mai simplă este

```
1 ubuntu@ubuntu:~$ > out.txt
```

[1] http://www.caliban.org/bash/index.shtml
[2] http://www.debian-administration.org/articles/316

- `.` – comanda este folosită pentru interpretarea unui fișier (de obicei script) în shell-ul curent. Echivalentă este comanda **`source`**. Un exemplu de folosire este încărcarea configurațiilor din `/etc/bash_completion`:

```
ubuntu@ubuntu:~$ . /etc/bash_completion
```

sau

```
ubuntu@ubuntu:~$ source /etc/bash_completion
```

- **`cd`** – comanda folosită pentru schimbarea directorului curent
- **`echo`** – afișarea șirului de caractere primit ca parametru la ieșirea standard
- **`alias`/`unalias`** – configurarea/dezactivarea configurării unui alias pentru o comandă. Astfel, dacă folosim:

```
alias mygrep='grep --color=tty -d skip'
```

comanda **`mygrep`** (alias) va fi lansată ca o comandă **`grep`** cu argumentele precizate; se poate folosi ca alias chiar numele comenzii, în cazul în care utilizatorul dorește rularea implicită cu acele argumente:

```
alias grep='grep --color=tty -d skip'
```

De notat este faptul că o comandă poate fi implementată atât în shell (*built-in*) cât și prin intermediul unui executabil. Astfel de comenzi sunt **`echo`** și **`time`**. În aceste situații comanda executată implicit este comanda internă. Pentru execuția comenzii externe trebuie apelată calea completă. În exemplul de mai jos sunt prezentate cele două tipuri de rulări pentru comanda **`time`** (internă și externă):

```
razvan@anaconda:~/code$ time ls
asm  boot_hello  latex  stack_ovfl  static  tests

real    0m0.004s
user    0m0.000s
sys     0m0.000s

razvan@anaconda:~/code$ /usr/bin/time ls
asm  boot_hello  latex  stack_ovfl  static  tests
0.00user 0.00system 0:00.00elapsed 133%CPU (0avgtext+0avgdata 0
maxresident)k
0inputs+0outputs (0major+254minor)pagefaults 0swaps
```

Se observă că poate exista o deosebire între formatele de ieșire ale unei comenzi interne, respectiv externe.

De asemenea, mecanismele de ajutor folosite pentru comenzi (**`whereis`**, **`apropos`**, `--help`, **`man`**, **`info`**) funcționează doar pentru comenzi externe. Pentru comenzi interne va trebui consultată documentația de *Bash* (**`man bash`**, **`info bash`**) sau folosită comanda (internă) **`help`**:

```
razvan@anaconda:~/code$ help time
time: time [-p] PIPELINE
    Execute PIPELINE and print a summary of the real time, user CPU time
,
    [...]

razvan@anaconda:~/code$ help alias
```

```
7 alias: alias [-p] [name[=value] ... ]
8      'alias' with no arguments or with the -p option prints the list
9     [...]
```

O listă completă a comenzilor interne *Bash* găsiți în pagina de manual. (**man bash**/**info bash** sau online[1])

12.2.4 Execuția unei comenzi shell

După cum s-a precizat, comenzile sunt de două tipuri: interne și externe. Clasificarea este dată de modul în care comenzile se execută în cadrul shell-ului. Astfel, comenzile interne (*built-in*) sunt executate direct în cadrul shell-ului, altfel spus în cadrul procesului curent. De partea cealaltă, execuția unei comenzi externe înseamnă crearea unui proces fiu în shell-ul curent și înlocuirea imaginii procesului cu executabilul asociat comenzii. Mai multe detalii despre caracteristicile și crearea unui proces se pot găsi în secțiunea 5.2.

Un proces astfel creat va conține ca imagine executabilul asociat comenzii. De obicei, shell-ul va aștepta terminarea procesului curent după care va oferi promptul utilizatorului. Acest efect poate fi schimbat prin rularea procesului în fundal (*background*) cu ajutorul operatorului & (vezi secțiunea 5.3).

De reținut este faptul că, în cazul înlănțuirii mai multor comenzi (vezi secțiunea 5.5.1), se creează un proces pentru fiecare comandă. Astfel, la rularea comenzii:

```
1 ubuntu@ubuntu:~$ ls -l | sort
```

se vor crea două procese: unul generat din executabilul **/bin/ls** iar altul generat din executabilul **/usr/bin/sort**. Shell-ul este procesul părinte al celor două procese și va asigura comunicația între acestea (ieșirea comenzii **ls -l** va fi redirectată către intrarea comenzii sort).

Figura 12.1 ilustrează grafic execuția comenzii de mai sus.

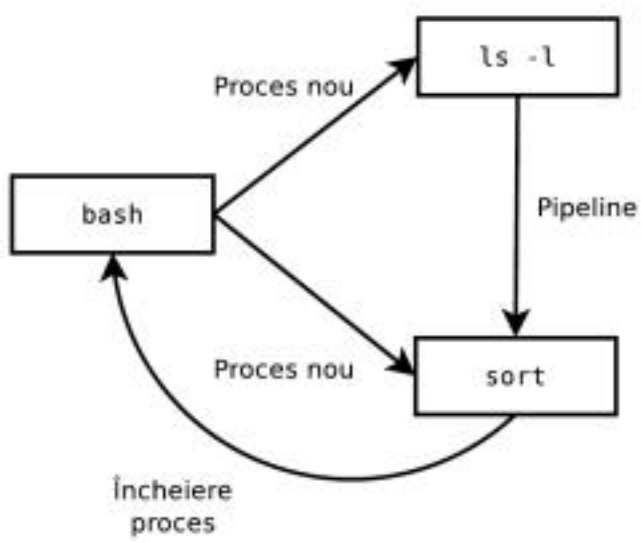


Figura 12.1: Execuția unei comenzi în shell

[1] http://www.gnu.org/software/bash/manual/bashref.html#SEC55

12.3 Scripturi shell

După cum am specificat, un script shell este un fișier text (un program) conținând comenzi shell, variabile, structuri de control și funcții cu rolul de rezolvare a unei probleme. Un script shell este interpretat de shell. Interpretarea înseamnă parcurgerea fiecărei înlănțuiri de comenzi sau structuri de control și realizarea sarcinilor descrise de aceasta.

Ceea ce separă un script shell de limbajele interpretate obișnuite este folosirea directă a comenzilor shell, a facilităților de înlănțuire a comenzilor puse la dispoziție și a fișierelor de configurare specifice sistemului.

Comenzile componente ale unui script shell pot fi executate manual la promptul shell-ului cu aceleași rezultate ca în cazul rulării scriptului. Astfel, liniile următoare dintr-un script shell:

```
1 if test 10 -gt 4; then
2   echo "mai mare"
3 else
4   echo "mai mic"
5 fi
```

pot fi executate direct în linia de comanda:

```
1 razvan@anaconda:~$ if test 10 -gt 4; then echo "mai mare"; else echo "mai
   mic"; fi
2 mai mare
```

sau astfel:

```
1 razvan@anaconda:~$ if test 10 -gt 4
2 > then
3 > echo "mai mare"
4 > else
5 > echo "mai mic"
6 > fi
7 mai mare
```

În ultimul exemplu, shell-ul oferă utilizatorului un miniprompt (>) pentru a indica acestuia că nu a finalizat comanda. În cazul de față comanda este `if`. Detalii despre `if` și despre efectul acestei bucăți de script se regăsesc în secțiunea 12.4.3.

12.3.1 Cel mai simplu script shell

Pentru a păstra tonul tutorialelor de limbaje de programare, cel mai simplu script shell pe care îl vom descrie va fi cel care va afișa mesajul "Hello, World!" la ieșirea standard. Programul este prezentat mai jos:

```
1 echo "Hello, World!"
2 exit 0
```

Programul este foarte simplu. Folosim comanda **`echo`** pentru a afișa un șir de caractere la ieșirea standard și comanda **`exit`** pentru ca programul să se termine cu succes.

Folosirea comenzii **exit** este opțională; un script shell va întoarce în mod implicit 0 dacă se ajunge la sfârșitul programului. Astfel, scriptul nostru shell poate fi scris într-o singură linie:

```
1 echo "Hello, World!"
```

Rularea unui script shell

Rularea unui script shell se realizează la fel ca în cazul unui limbaj interpretat prin transmiterea sa ca parametru interpretorului:

```
1 razvan@anaconda:~/uso/scripting$ bash hw.bash
2 Hello, World!
```

Scriptul shell poartă denumirea de `hw.bash`. Se recomandă ca scripturile shell Bash să aibă extensia `.sh` sau `.bash`, cu toate că în lumea Unix extensia poate fi omisă. Se observă afișarea mesajului "Hello, World!" la ieșirea standard.

În spatele acestei execuții, shell-ul generează un nou proces care interpretează conținutul scriptului primit ca parametru.

#! – shebang

Mecanismul uzual de execuție a unui shell script este acela prin care se execută un program obișnuit:

```
1 ubuntu@ubuntu:~$ ./hw.bash
```

Totuși, pentru a putea realiza acest lucru avem de îndeplinit două precondiții:

1. scriptul trebuie să fie executabil
2. în momentul execuției trebuie să se cunoască ce interpretor va fi folosit (`/bin/bash` pentru Bash, `/usr/bin/python` pentru Python, `/usr/bin/perl` pentru Perl etc.)

Primul pas se realizează foarte simplu. Se adaugă drepturi de execuție folosind comanda chmod:

```
1 razvan@anaconda:~/uso/scripting$ chmod a+x hw.bash
2
3 razvan@anaconda:~/uso/scripting$ ls -l hw.bash
4 -rwxr-xr-x 1 razvan razvan 42 Aug 31 18:46 hw.bash
```

Pentru realizarea celui de-al doilea pas se folosește perechea de caractere "`#!`" (diez și semnul exclamării) denumită și *shebang*. Această pereche de caractere este urmată de calea completă către programul care va interpreta scriptul. Linia care conține shebang și calea completă către interpretor este prima linie din script. În momentul în care scriptul este lansat în execuție, shell-ul parcurge prima linie și folosește interpretorul specificat pentru a executa scriptul. Exemple de linii *shebang* sunt:

- `#!/bin/bash` – executarea scriptului folosind Bash
- `#!/usr/bin/perl` – executarea scriptului folosind Perl

- `#!/usr/bin/php` – executarea scriptului folosind PHP
- `#!/usr/bin/python` – executarea scriptului folosind Python

Astfel, după adăugarea drepturilor de execuție și a liniei *shebang* scriptului nostru:

```
1 razvan@anaconda:~/uso/scripting$ cat hw.bash
2 #!/bin/bash
3
4 echo "Hello, World!"    # afisare mesaj
5 exit 0
```

îl putem executa:

```
1 razvan@anaconda:~/uso/scripting$ ./hw.bash
2 Hello, World!
```

Un script shell nu necesită prezența liniei *shebang*. În mod implicit un script este interpretat folosind **`/bin/bash`**. Se recomandă, totuși, prezența acestei linii pentru a asigura consistența.

În prezent, distribuția Debian și distribuțiile derivate (precum Ubuntu, Kubuntu etc.) folosesc ca shell *Bash*, iar ca interpretor implicit de script-uri shell *Dash*. Interpretorul de script-uri shell implicit este **`/bin/sh`** (link simbolic către *Dash*).

Dash este mult mai rapid și mai eficient din punct de vedere al consumului de resurse dar este și mai limitat ca și sintaxă, comparativ cu *Bash*. Așadar, atunci când modificați scripturi existente în sistem care au în linia *shebang* ca interpretor **`/bin/sh`**, luați în considerare că sintaxa este limitată[1] comparativ cu *Bash*.

12.3.2 Comentarii într-un script shell

Comentariile într-un script shell se realizează cu ajutorul caracterului `#`. Un comentariu începe de la apariția `#` până la sfârșitul liniei, asemănător cu `//` din C++. Exemple de comentarii sunt:

```
1 # comentariu
2 echo "hello"  # comentariu
3 # echo "hello"  (comanda comentata)
```

12.3.3 Comenzi simple shell

În mod evident, scriptul shell prezentat anterior nu este un script shell în adevăratul sens al cuvântului. Aceasta deoarece același efect poate fi obținut prin rularea simplă a comenzii echo urmată de argumentul "Hello, World!" la promtpul shell-ului. Un script shell va conține mai multe comenzi care vor fi interpretate într-o singură instanță în momentul execuției scriptului.

Scripturile shell folosesc de multe ori comenzi interne. Vom prezenta, în continuare, o parte din cele mai folositoare comenzi interne:

[1] https://wiki.ubuntu.com/DashAsBinSh

echo

Comanda **echo** permite afișarea șirului primit ca parametru la ieșirea standard a terminalului:

```
razvan@anaconda:~$ echo "test"
test
```

Opțiuni utile pentru **echo** sunt:

- `-n`: dezactivează afișarea implicită a unui caracter newline la sfârșitul liniei:

```
razvan@anaconda:~$ echo -n "test"
testrazvan@anaconda:~$
```

- `-e`: permite interpretarea caracterelor speciale:

```
razvan@anaconda:~$ echo -ne "test\n\t\ttest2\n"
test
                test2
```

printf

Comanda **printf** permite, ca și **echo**, afișarea unui șir de caractere la ieșirea standard. Permite, însă, la fel ca funcția `printf` din C, formatarea șirului de afișare:

```
razvan@anaconda:~$ printf "%s    %03d\n" "test" 20
test    020
```

Se observă similaritatea cu funcția printf din C: primul argument este șirul de formatare iar celelalte sunt argumentele de afișat la ieșirea standard.

exit

Comanda **exit** permite întreruperea execuției scriptului. Comanda poate primi un argument care specifică valoarea de retur. Această valoare este implicit 0.

export

Comanda **export** permite exportarea unei variabile (sub forma unei variabile de mediu) din scriptul curent în shell. Astfel, după ce shell-ul a încheiat interpretarea scriptului, variabila va putea fi folosită direct din shell. Pentru mai multe detalii în legătură cu variabile în script-uri consultați secțiunea 12.4.1.

set

Comanda **set** este utilizată pentru configurarea și personalizarea shell-ului. O opțiune utilă este opțiunea `-x` care permite afișarea comenzilor unui script shell pe măsură ce acestea sunt executate.

read

Comanda **read** este folosită pentru a citi date de la intrarea standard și pentru stocarea acestora într-o variabilă. În exemplul de mai jos, se dorește stocarea datelor introduse de utilizator în variabila `a`:

```
1 razvan@anaconda:~/uso/scripting$ read a
2 test_msg
3
4 razvan@anaconda:~/uso/scripting$ echo $a
5 test_msg
```

Se observă că variabila conține șirul *test_msg* introdus de utilizator.

12.3.4 One liners

Multe din problemele ce necesită utilizarea facilităților shell pot fi realizate prin intermediul unei simple linii. S-a observat că nu este necesar (și nici eficient) să utilizăm un script shell doar pentru afișarea unui mesaj precum "Hello, World!\n" la ieșirea standard.

Folosind redirectări sau înlănțuirea comenzilor se pot obține efecte imediate într-o singură linie de shell. Astfel de comenzi combinate poartă numele de **one liners**.

Redirectarea intrării/ieșirii unei comenzi

Redirectarea unei comenzi se oferă la posibilitatea de substituire a intrării, ieșirii sau erorii standard a unei comenzi cu un fișier. Astfel, redirectarea intrării standard înseamnă că o comandă va citi informații dintr-un fișier în locul citirii de la intrarea standard. Redirectarea ieșirii standard înseamnă că rezultatul comenzii se va regăsi într-un fișier de ieșire.

Pentru redirectarea intrării standard se folosește operatorul < (caracterul "mai mic") urmat de numele fișierul din care se realizează redirectarea. În cazul redirectării ieșirii standard, operatorul folosit este > (mai mare). Informații despre redirectarea intrării și ieșirii standard se regăsesc și în secțiunea 4.4.

Vom exemplifica redirectarea folosind utilitarul **cat**. Comanda **cat** realizează, în mod implicit, afișarea conținutului intrării standard la ieșirea standard. Un exemplu de rulare este prezentat în continuare:

```
1 razvan@anaconda:~$ cat
2 mesaj1
3 mesaj1
4 mesaj2
5 mesaj2
```

Pentru încheierea introducerii de informații de la intrarea standard trebuie folosită combinația `CTRL-D`, însemnând sfârșit de fișier (*end of file*). Se observă că ceea ce introduce utilizatorul la intrarea standard se afișează la ieșirea standard.

Dacă vom redirecta un fișier la intrarea standard, efectul va fi afișarea conținutului fișierului la ieșirea standard:

```
razvan@anaconda:~/uso/scripting$ cat < hw.bash
#!/bin/bash

echo "Hello, World!"

exit 0
```

Acest lucru este echivalent cu transmiterea fișierului ca argument comenzii **cat**.

În mod similar, dacă redirectăm ieșirea standard într-un fișier, acesta va conține ceea ce a introdus utilizatorul la intrarea standard (se folosește `CTRL-D` pentru încheiere):

```
razvan@anaconda:~/uso/scripting$ cat > out.txt
introducem mesaj
dupa care apasam CTRL-D

razvan@anaconda:~/uso/scripting$ cat out.txt
introducem mesaj
dupa care apasam CTRL-D
```

În exemplul de mai sus, comanda **cat** a afișat fișierul `out.txt` prin transmiterea acestuia ca argument, echivalent cu redirectarea acestuia către intrarea standard a comenzii. Acest comportament se extinde și la alte comenzii de prelucrare de fișiere text (vezi secțiunea 12.5).

Un efect care poate părea surprinzător este folosirea simultană a redirectării intrării și ieșirii standard. În acest fel, conținutul fișierului de la intrare este copiat în fișierul de la ieșire. Efectul este identic cu folosirea comenzii **cp**:

```
razvan@anaconda:~/uso/scripting$ cat < hw.bash > new.bash

razvan@anaconda:~/uso/scripting$ cat new.bash
#!/bin/bash

echo "Hello, World!"

exit 0
```

Pentru redirectarea erorii standard se folosește operatorul `2>`. Aceasta are legătură cu descriptorul de fișier asociat: intrarea standard are asociat descriptorul de fișier `0`, ieșirea standard are asociat descriptorul de fișier `1` iar eroarea standard are asociat descriptorul de fișier `2`.

Astfel, dacă se doresc redirectate într-un fișier erorile și avertismentele emise de **gcc** se folosește operatorul `2>`:

```
razvan@anaconda:~/uso/scripting$ gcc hw.c 2> err_warn.txt

razvan@anaconda:~/uso/scripting$ cat err_warn.txt
hw.c: In function 'main':
hw.c:3: warning: incompatible implicit declaration of built-in function '
printf'

razvan@anaconda:~/uso/scripting$ gcc hw.c 2> /dev/null
```

În cel de-al doilea exemplu, s-a redirectat eroarea standard către `/dev/null`. Aceasta înseamnă că nu se dorește afișarea la eroarea standard sau într-un fișier a diverselor informații de avertisment afișate de **gcc**.

Se poate redirecta atât ieșirea standard cât și eroarea standard folosind operatorul `&>` (Atenție: acest operator nu funcționeaza în interpretorul shell *Dash*[1]). În exemplu de mai jos se redirectează atât ieșirea standard cât și eroarea standard a execuției comenzii **`strace`**:

```
1 razvan@anaconda:~/uso/scripting$ strace ls &> out_ls.txt
```

Operatorul >> Operatorul `>>` este utilizat pentru redirectarea ieșirii standard însă cu adăugarea (*append*) a informațiilor redirectate la sfârșitul fișierului. Echivalent există operatorul `2>>` pentru adăugarea informațiilor de la eroarea standard. În exemplul de mai jos scriem ieșirile comenzilor **`ls`**, **`ps`** și **`uptime`** într-un același fișier:

```
1  razvan@anaconda:~/uso/scripting$ ls > out.txt
2
3  razvan@anaconda:~/uso/scripting$ ps >> out.txt
4
5  razvan@anaconda:~/uso/scripting$ uptime >> out.txt
6
7  razvan@anaconda:~/uso/scripting$ cat out.txt
8  a.out
9  err_warn.txt
10 hw.bash
11 hw.c
12 new
13 new.bash
14 out.txt
15 out_ls.txt
16   PID TTY          TIME CMD
17 30624 pts/2    00:00:00 bash
18 31508 pts/2    00:00:00 ps
19  15:34:59 up 59 days, 39 min,  2 users,  load average: 0.37, 0.55, 0.61
```

Here documents; operatorul << Operatorul `<<` este folosit în ceea ce se cheamă **here document**. Un *here document* este folosit, de obicei, cu utilitare interactive. Formatul de utilizare este:

```
1 command <<word
```

Acest lucru va impune comenzii citirea informației aflată în continuarea comenzii de la intrarea standard până la întâlnirea cuvântului *word* în corpul scriptului. În exemplul de mai jos, se citește de la intrarea standard până la întâlnirea cuvântului *END*:

```
1  razvan@anaconda:~/uso/scripting$ cat here.bash
2  #!/bin/bash
3
4  cat <<END
5  mesaj simplu
6  ... pentru un here document ...
7
8  avantajul este ca nu trebuie sa folosesc multe comenzi echo
9  END
10
11 echo "s-a terminat"
12 razvan@anaconda:~/uso/scripting$ bash here.bash
```

[1] https://wiki.ubuntu.com/DashAsBinSh

```
mesaj simplu
... pentru un here document ...

avantajul este ca nu trebuie sa folosesc multe comenzi echo
s-a terminat
```

După cum se observă și din comentariile fișierului, un *here document* poate fi folosit în locul comenzii **echo**. Un exemplu util îl constituie generarea unei pagini web simple. O primă alternativă este folosirea comenzii **echo**:

```
razvan@ragnarok:~/uso/scripting$ cat echo.bash
#!/bin/bash

hostname="ragnarok"
name="razvan"

echo -e "<html>"
echo -e "\t<title>"
echo -e "\t\tPagina mea"
echo -e "\t</title>"
echo -e "\t<body>"
echo -e "\t\t<h3>Statia este $hostname</h3>"
echo -e "\t\t<h3>Numele meu este $name</h3>"
echo -e "\t\t<p>Data este $(date)</p>"
echo -e "\t</body>"
echo -e "</html>"

razvan@ragnarok:~/uso/scripting$ bash echo.bash
<html>
        <title>
                Pagina mea
        </title>
        <body>
                <h3>Statia este ragnarok</h3>
                <h3>Numele meu este razvan</h3>
                <p>Data este Tue Sep  4 23:23:05 EEST 2007</p>
        </body>
</html>
```

Se observă că este incomodă folosirea comenzii **echo**. O soluție este utilizarea unui *here document*:

```
razvan@ragnarok:~/uso/scripting$ cat here.bash
#!/bin/bash

hostname="ragnarok"
name="razvan"

cat <<EndOfHtml
<html>
        <title>
                Pagina mea
        </title>
        <body>
                <h3>Statia este $hostname</h3>
                <h3>Numele meu este $name</h3>
                <p>Data este $(date)</p>
        </body>
</html>
EndOfHtml
```

Informații suplimentare despre *here documents* se pot găsi online[1].

Here strings; operatorul <<< Operatorul <<< permite folosirea de *here strings*, adică redirectarea unui șir de caractere către intrarea standard a unei comenzi. În exemplul de mai jos, se folosește comanda **read** pentru a citi conținutul unui șir într-un set de variabile:

```
razvan@anaconda:~$ read a b c d <<< "alfa beta gamma delta "

razvan@anaconda:~$ echo $a $b $c $d
alfa beta gamma delta
```

Ca și în cazul operatorului <<, același rezultat poate fi obținut prin folosirea comenzii **echo**:

```
razvan@anaconda:~$ echo "alfa beta gamma delta" | read a b c d
```

Trunchierea unui fișier **Trunchierea unui fișier** înseamnă eliminarea conținutului acestuia: fișierul devine gol (dimensiune 0). Trunchierea unui fișier se realizează prin redirectarea conținutului `/dev/null` în fișier. Acest lucru se realizează folosind comanda:

```
razvan@anaconda:~/uso/scripting$ ls -l out.txt
-rw-r--r-- 1 razvan razvan 220 Sep  4 15:34 out.txt

razvan@anaconda:~/uso/scripting$ cat /dev/null > out.txt

razvan@anaconda:~/uso/scripting$ ls -l out.txt
-rw-r--r-- 1 razvan razvan 0 Sep  4 15:35 out.txt
```

Mai simplu, trunchierea unui fișier se realizează prin redirectarea unei comenzi care nu afișează nimic la ieșirea standard. O astfel de comandă este : (comanda care nu face nimic):

```
razvan@anaconda:~/uso/scripting$ : > out.txt
```

Şi mai simplu, trunchierea se realizează folosind comanda de mai jos:

```
razvan@anaconda:~/uso/scripting$ > out.txt
```

Comanda este echivalentă cu cea de mai sus, cu absența comenzii :.

Înlănțuirea comenzilor

Shell-ul pune la dispoziția utilizatorului operatori care permit înlănțuirea diverselor comenzi pentru obținerea de noi funcționalități. Dintre aceștia, cel mai cunoscut este **operatorul** | (*pipe*) care redirectează ieșirea unei comenzi la intrarea alteia. Detalii despre acest operator se găsesc și în capitolul 5.

[1] http://tldp.org/LDP/abs/html/here-docs.html

Operatorul | (pipe) Un exemplu de utilizare al operatorului | este căutarea unei informații într-un fișier. O comandă uzuală este:

```
ubuntu@ubuntu:~$ cat file.txt | grep keyword
```

Căutarea informației în fișier se realizează cu ajutorul comenzii **grep**, descrisă în secțiunea 12.5.7.

Un exemplu mai complex care utilizează o serie de comenzi prezentate în acest capitol este determinarea numărului de utilizatori unici care s-au autentificat în sistem într-o zi dată. Comenzile utilizate sunt **last**, **grep**, **cut**, **sort**, **uniq**, **wc**. Detalii despre aceste vor fi prezentate în secțiunile următoare.

Pentru început, vom folosi comanda **last** care afișează autentificările în sistem. Comanda **grep** este folosită pentru a extrage numai acele linii care conțin ziua dorită:

```
razvan@anaconda:~/uso/scripting$ last -30 | grep Mon
sergiu   pts/4        89.18.20.56      Mon Sep  3 23:36 - 23:46  (00:09)
razvan   pts/4        dhcp-204.cs.pub. Mon Sep  3 23:19 - 23:35  (00:15)
carpalex pts/0        86.121.140.140   Mon Sep  3 23:14 - 01:41  (02:27)
alexef   pts/0        89.120.196.59    Mon Sep  3 22:42 - 22:51  (00:08)
cojocar  pts/0        86.127.17.156    Mon Sep  3 19:50 - 19:51  (00:01)
ddvlad   pts/3        84.247.45.23     Mon Sep  3 16:00 - 16:01  (00:00)
root     pts/0        dhcp-204.cs.pub. Mon Sep  3 15:14 - 17:33  (02:18)
ddvlad   pts/0        84.247.45.23     Mon Sep  3 15:07 - 15:09  (00:01)
valentin pts/2        86.122.195.234   Mon Sep  3 13:47 - 02:43  (12:55)
alexef   pts/0        89.120.196.59    Mon Sep  3 13:30 - 15:04  (01:34)
razvan   pts/0        dhcp-204.cs.pub. Mon Sep  3 10:30 - 10:44  (00:13)
stefanb  pts/0        adsl-76-200-147- Mon Sep  3 10:05 - 10:17  (00:12)
stefanb  pts/0        adsl-76-200-147- Mon Sep  3 06:13 - 07:57  (01:43)
```

După obținerea autentificărilor în sistem vrem să extragem numai numele utilizatorilor. Pentru aceasta vom folosi comanda **cut**:

```
razvan@anaconda:~/uso/scripting$ last -30 | grep Mon | cut -d' ' -f1
sergiu
razvan
carpalex
alexef
cojocar
ddvlad
root
ddvlad
valentin
alexef
razvan
stefanb
stefanb
```

În acest moment avem numele utilizatorilor care s-au autentificat în sistem în ziua de *Luni*. Mai departe, va trebui să extragem o singură instanță a numelor care se repetă. Drept urmare vom folosi utilitarele **sort** și **uniq**:

```
razvan@anaconda:~/uso/scripting$ last -30 | grep Mon | cut -d' ' -f1 |
sort | uniq
alexef
carpalex
cojocar
ddvlad
```

```
6  razvan
7  root
8  sergiu
9  stefanb
10 valentin
```

Mai rămâne să afișăm numărul de autentificări. Acest lucru se realizează prin contorizarea liniilor obținute la pasul anterior, cu ajutorul utilitarului **wc**:

```
1  razvan@anaconda:~/uso/scripting$ last -30 | grep Mon | cut -d' ' -f1 |
   sort | uniq | wc -l
2  9
```

Comanda de mai sus poate da rezultate incorecte în cazul în care `Mon` mai apare altundeva în rezultatul furnizat de **`last`**. Dacă ar exista un utilizator `Monica`, atunci vor apărea și liniile ce descriu autentificarea în sistem a utilizatorului `Monica` (chiar dacă autentificarea a avut sau nu loc în ziua de luni). Această problemă va fi corectată în secțiunea 12.11.1.

Operatorii `;`, `||` și `&&` Operatorii `;`, `||` și `&&` de înlănțuire a comenzilor țin cont de valoarea întoarsă de comanda anterioară. Astfel:

- `comm1 ; comm2` – comanda **comm2** se execută după execuția comenzii **comm1**
- `comm1 || comm2` – comanda **comm2** se execută în cazul în care **comm1** se întoarce cu o valoare nenulă
- `comm1 && comm2` – comanda **comm2** se execută în cazul în care **comm1** se încheie cu succes (întoarce 0)

Operatorul `;` este echivalent cu execuția secvențială a fiecărei comenzi.

Operatorul `&&` este folosit când **comm2** depinde de execuția cu succes a comenzii **comm1**. O situație este compilarea din surse și instalarea unei aplicații:

```
1  root@ubuntu:~# make && make install
```

12.4 Programarea shell

După prezentarea câtorva comenzi simple și a operatorilor shell, putem pătrunde în aspectele de programare a unui shell script. După cum s-a precizat, shell-ul pune la dispoziția utilizatorului un limbaj de programare. Acesta conține, ca și alte limbaje de programare, variabile, funcții, structuri de ciclare, instrucțiuni de decizie. Combinate cu celelalte facilități oferite de shell (utilizarea de comenzi deja implementate, înlănțuirea comenzilor, redirectarea), programarea shell devine un instrument puternic la îndemâna utilizatorului.

12.4.1 Variabile

O variabilă are asociat un nume și o valoare. La fel ca în C, numele variabilelor este un șir de caractere ce poate conține litere, cifre sau caracterul _ (*underscore*) și trebuie

să înceapă cu literă sau *underscore*. Exemple de nume de variabile sunt `a`, `a0b`, `a_b`, `AxB_01_D` etc.

Spre deosebire de limbaje precum C, variabilele în shell nu au un tip. Faptul că o variabilă este un număr sau un șir de caractere depinde de contextul în care este folosită. De asemenea, o variabilă poate fi definită oriunde. În mod obișnuit, unei variabile îi este asociată o valoare în momentul definirii. Exemple de inițializări sunt:

```
1 a=0
2 b="alfa"
3 c="a08ss"
4 d=230
```

Un lucru important de reținut este fragilitatea programării shell și constrângerile importante pe care le impune. Una dintre ele este faptul că nu poate exista spațiu înainte sau după semnul = din momentul inițializării unei variabile. Astfel, următoarele inițializări vor fi invalide:

```
1 a =0
2 b = "alfa"
3 d= 230
```

Valoarea unei variabile este referită cu ajutorul simbolului `$`. Astfel, dacă dorim afișarea valorii variabilelor definite mai sus folosim comanda:

```
1 razvan@anaconda:~$ echo $a $b $c $d
2 0 alfa a08ss 230
```

În consecință, dacă dorim inițializarea unei variabile la valoarea unei alteia vom folosi o construcție de forma:

```
1 razvan@anaconda:~$ echo $e
2 0
3
4 razvan@anaconda:~$ f="$b$c"
5
6 razvan@anaconda:~$ echo $f
7 alfaa08ss
```

În ultimul exemplu se observă că variabila `f` conține valorile concatenate ale celorlalte două variabile. De asemenea, se poate observa că ghilimelele nu schimbă sensul simbolului `$`.

Dacă se dorește ca o variabilă să fie un rezultat al unei operații aritmetice cu o altă variabilă se folosește operatorul de expandare aritmetică descris în secțiunea 12.7.1:

```
1 razvan@anaconda:~$ a=4
2
3 razvan@anaconda:~$ c=3
4
5 razvan@anaconda:~$ echo $(($a * $c + 1))
6 13
```

O facilitate importantă a shell-ului este posibilitatea stocării ieșirii unei comenzi într-o variabilă. Acest lucru se realizează prin intermediul construcției `$(command)`. În exemplul de mai jos, reținem în variabila `local_users_num` numărul de utilizatori care au directorul de bază în `/home`:

```
razvan@anaconda:~$ local_users_num=$(cat /etc/passwd | grep /home | wc -l
)

razvan@anaconda:~$ echo $local_users_num
52
```

Dacă vrem să contorizăm și utilizatorul `root`, adăugăm 1 la acea variabilă:

```
razvan@anaconda:~$ total_local=$(($local_users_num + 1))

razvan@anaconda:~$ echo $total_local
53
```

O greșeală frecventă în inițializarea unei variabile este folosirea de spații înainte sau după =. Evitați apariția unor astfel de greșeli în scripturile voastre.

Exemple

- afișarea ultimelor n adrese IP de la care s-au realizat conexiuni pentru un serviciu (inspecție de jurnale)

```
root@anaconda:/home/razvan/uso/scripting# cat -n service_ip.bash
     1  #!/bin/bash
     2
     3  ip_num=3
     4  log_file="/var/log/apache/access.log"
     5
     6  tail -n $ip_num $log_file | cut -d ' ' -f 1

root@anaconda:/home/razvan/uso/scripting# bash service_ip.bash
38.99.44.102
38.99.44.102
38.99.44.102
```

- Afișarea memoriei totale ocupate de primele 3 procese din sistem[1]

```
root@anaconda:/home/razvan/uso/scripting# cat -n mem_proc.bash
     1  #!/bin/bash
     2
     3  rss1=$(ps -e -o rss --sort=-rss | head -n 2 | tail -n 1)
     4  rss2=$(ps -e -o rss --sort=-rss | head -n 3 | tail -n 1)
     5  rss3=$(ps -e -o rss --sort=-rss | head -n 4 | tail -n 1)
     6
     7  total_rss_kb=$(($rss1 + $rss2 + $rss3))
     8
     9  echo "memorie totala ocupata de primele 3 procese:
$total_rss_kb KB"

root@anaconda:/home/razvan/uso/scripting# bash mem_proc.bash
memorie totala ocupata de primele 3 procese: 29088 KB
```

- Adăugarea unui utilizator în sistem în mod neinteractiv

```
root@anaconda:/home/razvan/uso/scripting# cat -n my_add_user.sh
     1  #!/bin/bash
```

[1] presupunem ca nu se modifică procesele pe parcursul execuției script-ului, astfel încât ordinea lor să se păstreze

```
 3      2
 4      3  #
 5      4  # Add a new user; invoke all necesary script and command for
    adding complete
 6      5  # user facilities
 7      6  #
 8      7
 9      8  username="newuser"
10      9  email="razvand@gmail.com"
11     10
12     11  GROUP_NAME="students"
13     12  GROUP_DIR="students"
14     13  QUOTA="250"             # size in megs
15     14
16     15  # add user
17     16  useradd -m -d /home/$GROUP_DIR/$username -g $GROUP_NAME -s /
    bin/bash $username
18     17
19     18  # create password
20     19  password=$(pwgen -N 1)
21     20
22     21  # add password - non-interractive mode
23     22  echo "$username:$password" | chpasswd
24     23
25     24  # set quota
26     25  setquota $username $(($QUOTA * 1024)) $(($QUOTA + 10) *
    1024)) $(($QUOTA * 10)) $(($QUOTA + 10) * 10)) -a
27     26
28     27  # send e-mail
29     28  mail -s "New Account" $email <<EndOfMsg
30     29  A new account has been created for you. Username is
    $username.
31
32     30  Your password is $password. Please use passwd to change it.
33
34     31
35     32  Your qota limit is $QUOTA MB.
36
37     33
38     34  Have a nice day!
39     35  EndOfMsg
40     36
41     37  exit 0
42
43  root@anaconda:/home/razvan/uso/scripting# ./my_add_user.sh
44
45  root@anaconda:/home/razvan/uso/scripting# cat /etc/passwd | tail -1
46  newuser:x:1051:1026::/home/students/newuser:/bin/bash
```

12.4.2 Caractere speciale shell

Shell-ul are un set de caractere rezervate, caractere ce au roluri bine definite într-un script.

Caracterul blank (spațiu)

Caracterul blank este folosit pentru a separa argumentele unei comenzi de comandă și între ele. Astfel, în exemplul

```
ubuntu@ubuntu:~$ ls -l mydir
```

avem comanda **`ls`** cu două argumente: `-l` și `mydir`. Se poate întâmpla să avem un argument care conține caractere blank. Spre exemplu, avem directorul `my dir`. Pentru afișarea conținutului acestui director, nu putem folosi comanda:

```
ubuntu@ubuntu:~$ ls my dir
```

întrucât shell-ul ar considera două argumente transmise comenzii **`ls`**. Astfel, s-ar încerca afișarea conținutului directorului `my` și apoi a directorului `dir`. Pentru a preîntâmpina acest efect, caracterul blank trebuie "citat", adică trebuie folosit ca un caracter obișnuit. Pentru aceasta există trei soluții:

1. folosirea caracterului special *ghilimele* pentru "înglobarea" numelui de director ce conține blank:

```
ubuntu@ubuntu:~$ ls "my dir"
```

2. folosirea caracterului special *apostrof*, la fel cum s-a folosit ghilimele:

```
ubuntu@ubuntu:~$ ls 'my dir'
```

3. folosirea caracterului special *backslash*, care dezactivează caracterul special de după el

```
ubuntu@ubuntu:~$ ls my\ dir
```

Despre caracterele speciale *ghilimele*, *apostrof* și *backslash* se va discuta în continuare.

Caracterul $ (dolar)

Unul dintre acestea este caracterul `$` care poate fi folosit pentru expandarea unei variabile (a valorii acesteia), a unei comenzi sau expandare aritmetică.

Expandarea unei variabile se referă la determinarea valorii acesteia:

```
razvan@anaconda:~$ a=3

razvan@anaconda:~$ b=mesaj

razvan@anaconda:~$ echo $a $b
3 mesaj
```

Expandarea unei comenzi se refera la reținerea rezultatului execuției comenzii:

```
razvan@anaconda:~$ num_dirs=$(ls -l | wc -l)

razvan@anaconda:~$ echo $num_dirs
21
```

Expandare aritmetică se referă la executarea de calcule aritmetice. Fără folosirea operatorului de expandare aritmetică, operatorii aritmetici ar fi considerați caractere simple:

```
razvan@anaconda:~$ echo 2+3+4
2+3+4

razvan@anaconda:~$ echo $((2+3+4))
9
```

Mai multe informații despre utilizarea caracterului `$` și despre expandare sunt prezentate în secțiunea 12.7.

Caracterul `$` este un caracter special. Acest lucru înseamnă că nu poate fi folosit cu semnificația de caracter direct când este urmat de un alt caracter care impune o expandare. Dacă dorim afișarea șirului `$a`, nu putem folosi comanda **`echo $a`**, pentru că ar afișa valoarea variabilei `a`:

```
razvan@anaconda:~$ echo $a
3
```

Pentru aceasta trebuie folosit caracterul *backslash* sau *apostrof*.

Folosirea ghilimelelor nu conduce la folosirea semnificației de caracter pentru `$`:

```
razvan@anaconda:~$ echo "$a"
3

razvan@anaconda:~$ echo '$a'
$a

razvan@anaconda:~$ echo \$a
$a
```

Caracterul " (ghilimele)

Un alt caracter special este " (ghilimele). Acest caracter este folosit pentru a defini șiruri de caractere:

```
$ a="sir"
$ b="acest sir"
```

Caracterul ghilimele este folosit pentru pastrarea semnificației de caracter pentru caracterele speciale: *blank*, `(`, `)`, `&`, `|` etc. Excepția o constituie caracterul `$` care își păstrează semnificația în cazul expansiunii:

```
razvan@anaconda:~$ c=3

razvan@anaconda:~$ b=a

razvan@anaconda:~$ echo "c = $c; b = $b"
c = 3; b = a
```

Pentru afișarea unui caracter ghilimele putem folosi *backslash* sau *apostrof*:

```
razvan@anaconda:~$ echo "simbolul ghilimele (\")"
simbolul ghilimele (")

razvan@anaconda:~$ echo 'simbolul ghilimele (")'
simbolul ghilimele (")
```

Caracterul \ (backslash)

S-a observat însă că nu putem afișa caracterul `$`. Comanda de mai jos afișează valoarea variabilei `a`:

```
1 razvan@anaconda:~$ echo $a
2 3
```

Shell-ul interpretează caracterul `$` ca inițiator al unei expandări. Dacă dorim afișarea șirului `$a`, nu putem folosi construcția de mai sus, întrucât se va încerca expandarea variabilei `a`. Pentru a realiza acest lucru vom folosi caracterul `\`(*backslash*). Acest caracter este caracterul de citarea și are rolul de a păstra semnificația caracterului de după el. Citare înseamnă folosirea unui caracter cu semnificația literală.

Astfel, dacă dorim afișarea șirului `$a`, folosim comanda:

```
1 razvan@anaconda:~$ echo \$a
2 $a
```

Caracterul `\` poate fi folosit și pentru citarea caracterului `"`:

```
1 razvan@anaconda:~$ echo \"
2 "
```

Tot `\` este folosit pentru afișarea `\`:

```
1 razvan@anaconda:~$ echo \\
2 \
```

Caracterul ' (apostrof)

Caracterul `'` are un rol similar cu cel al caracterului `"`. Este folosit pentru descrierea de șiruri și forțează caracterele speciale să-și păstreze semnificația. Deosebirea între ' și " este că primul este mult mai puternic. Astfel, orice caractere speciale care apar între două caractere apostrof își păstrează semnificația:

```
1 razvan@anaconda:~$ echo ' " ) ; & $ \ '
2  " ) ; & $ \
```

Alte caractere speciale

Alte caractere cu rol special în shell sunt cele folosite de operatorii shell: `>`, `<`, `|`, `&`, `;`, `(`, `)` sau cele folosite pentru expresii regulate în shell: `{`, `}`, `*`, `+`, `?`, `[`, `]`. Toate aceste caractere își păstrează semnificația dacă sunt folosite între ghilimele sau apostroafe. Mai multe detalii despre caracterele folosite pentru expresii regulate în shell se regăsec în secțiunea 12.7.2.

Dacă, spre exemplu, un utilizator dorește afișarea tuturor pachetelor instalate care încep cu litera `n`, va trebui să utilizeze comanda:

```
1 ubuntu@ubuntu:~$ dpkg -l n*
```

Comanda de mai jos nu va funcționa dacă există o intrare în directorul curent care începe cu litera `n` din cauza expandării expresiei regulate de către shell, fără a fi transmis către utilitarul **`dpkg`**:

```
razvan@anaconda:~/uso/scripting$ ls
hw.bash  new

razvan@anaconda:~/uso/scripting$ dpkg -l n*
No packages found matching new.
```

Soluția este citarea caracterului *:

```
razvan@anaconda:~/uso/scripting$ dpkg -l 'n*'
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err:
uppercase=bad)
||/ Name           Version        Description
+++-==============-==============-============================================

ii  nano           2.0.2-1etch1   free Pico clone with some new features
un  nano-tiny      <none>         (no description available)
un  nas            <none>         (no description available)
un  nas-lib        <none>         (no description available)
pn  nautilus       <none>         (no description available)
pn  nautilus-cd-bu <none>         (no description available)
ii  nautilus-data  2.14.3-11      data files for nautilus
```

12.4.3 Instrucțiuni de decizie

Instrucțiunile de decizie folosite în scripturile shell sunt `if` și `case`. O instrucțiune de decizie permite testarea unei condiții și executarea unei acțiuni sau a alteia conform condiției.

if

Vom analiza instrucțiunea `if` prin următorul exemplu: dacă utilizatorul local este `root` atunci se afișează un mesaj, dacă utilizatorul local este `razvan` se afișează un alt mesaj, iar dacă utilizatorul local este altcineva, se afișează un mesaj specific:

```
razvan@anaconda:~/uso/scripting$ cat -n if_user.bash
     1  #!/bin/bash
     2
     3  user=$(whoami)
     4
     5  if test $user = "root"; then
     6          echo "Bow before me for I am root."
     7  elif test $user = "razvan"; then
     8          echo "I am no humble user."
     9  else
    10          echo "I am but a humble user."
    11  fi
12
razvan@anaconda:~/uso/scripting$ bash if_user.bash
I am no humble user.
```

Opțiunea `-n` a comenzii **`cat`** afișează și numărul liniei. Sintaxa `if` este următoarea:

```
1 if conditie1; then
2   actiune conditie1 indeplinita
3 elif conditie2; then
4   actiune conditie2 indeplinita
5 else
6   actiune conditie neindeplinita
7 fi
```

Cuvinte cheie predefinite sunt `if`, `then`, `elif`, `else`, `fi`. *conditie1*, respectiv *conditie2* sunt condiții care sunt testate. În cazul în care o conditie este îndeplinită se execută acțiunea asociată.

După cum se vede din exemplu (liniile 5 și 7), condițiile sunt exprimate cu ajutorul utilitarului test. Acesta interpretează condiția și întoarce 0 dacă este îndeplinită sau 1 dacă nu:

```
1 razvan@anaconda:~/uso/scripting$ test "alfa" = alfa
2
3 razvan@anaconda:~/uso/scripting$ echo $?
4 0
5
6 razvan@anaconda:~/uso/scripting$ test "alfa" = alf
7
8 razvan@anaconda:~/uso/scripting$ echo $?
9 1
```

Variabila `$?` este o variabilă predefinită care deține valoarea de retur a ultimei expresii.

Același efect poate fi obținut cu ajutorul operatorului de test al shell-ului `[ .. ]`:

```
1 razvan@anaconda:~/uso/scripting$ [ "alfa" = alfa ]
2
3 razvan@anaconda:~/uso/scripting$ echo $?
4 0
5
6 razvan@anaconda:~/uso/scripting$ [ "alfa" = alf ]
7
8 razvan@anaconda:~/uso/scripting$ echo $?
9 1
```

Este obligatoriu un caracter blank (spațiu) după [și înainte de] la operatorul de test de condiție din shell.

Condiția folosită `if` nu trebuie să fie neapărat o condiție de test. Se poate folosi orice construcție care poate fi evaluată la un număr. Dacă acel număr este 0 condiția este îndeplinită și se execută acțiunea corespunzătoare, altfel nu. În exemplu următor, se afișează un mesaj corespunzător doar dacă utilizatorul indicat de variabila `user` are directorul de bază în `/home` (nu este utilizator de sistem):

```
1 razvan@anaconda:~/uso/scripting$ cat -n home_user.bash
2      1  #!/bin/bash
3      2
4      3  user="mihai"
5      4
6      5  if grep "$user" /etc/passwd | grep "/home"; then
7      6          echo "User $user is not homeless."
8      7  else
9      8          echo "User $user is homeless."
```

```
10      9  fi
11
12  razvan@anaconda:~/uso/scripting$ bash home_user.bash
13  amihaiuc:x:1011:1011:Alex Mihaiuc,,,:/home/amihaiuc:/bin/bash
14  mihai:x:1014:1014:Mihai Dobrescu,,,:/home/mihai:/bin/bash
15  mihaif:x:1023:1023:Mihai Florian,,,:/home/mihaif:/bin/bash
16  User mihai is not homeless.
```

echo -n "test" Se observă că **if** testează ieșirea comenzii compuse

```
1  grep "$user" /etc/passwd | grep "/home"
```

care verifică dacă există utilizatorul `mihai` în sistem și dacă acest utilizator are directorul de bază în `/home` (linia 5). Întrucât acesta există, afișează mesajul "User mihai is not homeless.".

Totuși, comanda **grep** afișează și liniile găsite, lucru care este deranjant în perspectiva funcționalității scriptului. Soluția este redirectarea ieșirii comenzii **grep** către `/dev/null`.

```
1   razvan@anaconda:~/uso/scripting$ cat -n home_user.bash
2        1  #!/bin/bash
3        2
4        3  user="mihai"
5        4
6        5  if grep "mihai" /etc/passwd | grep "/home" > /dev/null ; then
7        6          echo "User $user is not homeless."
8        7  else
9        8          echo "User $user is homeless."
10       9  fi
11
12  razvan@anaconda:~/uso/scripting$ bash home_user.bash
13  User mihai is not homeless.
```

Condiții de test Atât `test` cât și operatorul de condiție shell `[..]` analizează o condiție și întorc 0 dacă este îndeplinită sau 1 dacă nu este îndeplinită. În exemplele prezentate de anterior condiția a fost `sir1 = sir2`. Există mai multe condiții, o parte din care sunt prezentate în tabelul de mai jos:

Mai multe condiții pot fi folosite simultan folosind operatorii `-a` și `-o` cu sintaxele respectiv `conditie1 -a conditie2` și `conditie1 -o conditie2`.

Câteva exemple de folosire a condițiilor de mai sus sunt prezentate în continuare:

```
1   razvan@anaconda:~/uso/scripting$ test -f a.txt
2
3   razvan@anaconda:~/uso/scripting$ test -s a.txt
4
5   razvan@anaconda:~/uso/scripting$ test a.txt -ot b.txt
6
7   razvan@anaconda:~/uso/scripting$ test 3 -eq 4
8
9   razvan@anaconda:~/uso/scripting$ test -z ""
10
11  razvan@anaconda:~/uso/scripting$ test -z "a"
12
13  razvan@anaconda:~/uso/scripting$ test "a" = "b"
14
```

```
15 razvan@anaconda:~/uso/scripting$ test -f a.xt -a 3 -eq 4
```

Negarea unui condiții se face cu ajutorul operatorului `!`:

```
1 razvan@anaconda:~$ if ! test 4 -gt 5; then echo "4 nu e mai mare ca 5";
  else echo "4 e mai mare ca 5"; fi
2 4 nu e mai mare ca 5
```

Operatorul `((...))` de expansiune aritmetică poate fi folosit pentru evaluarea unor condiții într-un format asemănător C:

```
1 razvan@anaconda:~/uso/scripting$ if ((1 < 4)); then echo "adevarat"; fi
2 adevarat
```

Mai multe detalii despre operatorul `((...))` se găsesc în secțiunea 12.7.1.

case

Instrucțiunea `case` permite selectarea între mai multe șabloane de șiruri de expresii regulate. În exemplul de mai jos se verifică dacă un șir începe cu o literă mică, o literă mare sau cu un număr:

```
1  razvan@anaconda:~/uso/scripting$ cat -n case_begin.bash
2       1  #!/bin/bash
3       2
4       3  var="Alfa"
5       4
6       5  case $var in
7       6          [a-z]*) echo "litera mica";;
8       7          [A-Z]*) echo "litera mare";;
9       8          [0-9]*) echo "cifra";;
10      9          *) echo "altceva";;
11     10  esac
12
13 razvan@anaconda:~/uso/scripting$ bash case_begin.bash
14 litera mare
```

Sintaxa pentru `case`, observabilă și în exemplu, este:

```
1 case valoare_variabila in
2   sablon1) actiuni;;
3   sablon2) actiuni;;
4   ...
5
6 esac
```

Cuvinte rezervate sunt `case`, `in`, `esac`. Se verifică dacă `valoare_variabila` se potrivește cu unul dintre șabloanele ulterioare. În momentul găsirii unei potriviri se execută acțiunile asociate. Acțiunile asociate șablonului se încheie cu `;;` (două simboluri "punct și virgulă").

În exemplul de mai sus, variabila `var` se inițializează la șirul "`Alfa`" (linia 3). După aceasta se verifică dacă variabila începe cu literă mică (linia 6), cu literă mare (linia 7) sau cu cifră (linia 8). Şabloanele sunt date, de obicei, de expresii regulate. Mai multe detalii despre expresii regulate se găsesc în secțiunea 12.7.2.

Caracterul `*` înseamnă orice și este folosit ca echivalent pentru cuvântul cheie `default` din C.

12.4.4 Cicluri în shell

Bash pune la dispoziția utilizatorului trei instrucțiuni de ciclare cu funcționalitate similară dar sintaxă diferită: `for`, `while` și `do/until`. Vom exemplifica sintaxa și modul de utilizare a acestor comenzi pe un exemplu simplu: calculul sumei primelor 10 numere naturale.

Instrucțiunea `for`

Scriptul shell pentru calculul sumei primelor 10 numere naturale este următorul:

```
razvan@anaconda:~/uso/scripting$ cat -n for10.bash
     1  #!/bin/bash
     2
     3  sum=0
     4  for i in 1 2 3 4 5 6 7 8 9 10; do
     5          sum=$(($sum + $i))
     6  done
     7
     8  echo "Suma este: $sum"

razvan@anaconda:~/uso/scripting$ bash for10.bash
Suma este: 55
```

După cum se vede, sintaxa `for` este

```
for var in lista; do
  actiuni
done
```

sau

```
for var in lista
do
  actiuni
done
```

Varibila `var` este folosită pentru a parcurge lista element cu element. Cuvinte rezervate sunt `for`, `in`, `do` și `done`.

Se parcurg elementele listei (linia 4) și se execută acțiunile precizate (linia 5). În cazul de față (linia 5), folosind operatorul de expansiune aritmetică `$((...))` se incrementează suma cu valoarea lui `i`. Ca orice alte facilități shell, `for` poate fi folosit direct în linia de comandă:

```
razvan@anaconda:~/uso/scripting$ for i in 1 2 3 4; do echo $i; done
1
2
3
4
```

Acest lucru se poate extinde și la scriptul de mai sus cu pierderea parțială a clarității:

```
razvan@anaconda:~/uso/scripting$ sum=0; for i in 1 2 3 4 5 6 7 8 9 10; do
 sum=$(($sum+$i)); done; echo "Suma este: $sum"
Suma este: 55
```

sau:

```
razvan@anaconda:~$ sum=0

razvan@anaconda:~$ for i in 1 2 3 4 5 6 7 8 9 10; do
> sum=$(($sum + $i))
> done

razvan@anaconda:~$ echo $sum
55
```

Lista poate conține orice fel de element, fie număr fie șir de caractere:

```
razvan@anaconda:~/uso/scripting$ for i in abc def ghi; do echo $i; done
abc
def
ghi
```

Separatorul implicit pentru listă este caracterul *blank* (spațiu). Dacă un șir conține *blank*, acesta trebuie citat:

```
razvan@anaconda:~/uso/scripting$ for i in "abc def" ghi; do echo $i; done
abc def
ghi

razvan@anaconda:~/uso/scripting$ for i in 'abc def' ghi; do echo $i; done
abc def
ghi

razvan@anaconda:~/uso/scripting$ for i in abc\ def ghi; do echo $i; done
abc def
ghi
```

Comanda `seq` Revenind la calculul sumei primelor 10 numere naturale se observă lipsa de scalabilitate a specificării manuale a elementelor listei. Acest aspect poate fi automatizat prin folosirea comenzii **`seq`**. Aceasta permite generarea unei liste de numere într-un interval dat:

```
razvan@anaconda:~/uso/scripting$ seq 1 4
1
2
3
4

razvan@anaconda:~/uso/scripting$ seq 1 2 10
1
3
5
7
9
```

Cu două argumente, `seq` afișează elementele cuprinse între cele două limite. Cu trei argumente, argumentul din mijloc (al doilea) este folosit ca pas de incrementare.

Dacă, spre exemplu, am dori calculul sumei primelor 100 de numere naturale am folosi urmatorul script shell:

```
razvan@anaconda:~/uso/scripting$ cat -n for10_seq.bash
```

```
     1  #!/bin/bash
     2
     3  sum=0
     4  for i in $(seq 1 100); do
     5          sum=$(($sum + $i))
     6  done
     7
     8  echo "Suma este: $sum"

razvan@anaconda:~/uso/scripting$ bash for10_seq.bash
Suma este: 5050
```

O altă soluție este folosirea expandării aritmetice ca în exemplul de mai jos:

```
razvan@anaconda:~/uso/scripting$ cat -n for10_exp.bash
     1  #!/bin/bash
     2
     3  sum=0
     4  for ((i = 1; i <= 100; i++)); do
     5          sum=$(($sum + $i))
     6  done
     7
     8  echo "Suma este: $sum"

razvan@anaconda:~/uso/scripting$ bash for10_exp.bash
Suma este: 5050
```

Se observă similaritatea cu sintaxa C pentru instrucțiunea `for`.

Lucrul cu sistemul de fișiere Folosind `for` putem parcurge intrările dintr-un director al sistemului de fișier. O emulare a comenzii `ls` este următorul script shell:

```
razvan@anaconda:~/uso/scripting$ cat -n for_ls.bash
     1  #!/bin/bash
     2
     3  n=0
     4  for i in *; do
     5          n=$(($n + 1))
     6          if test -f "$i"; then
     7                  echo "f - $i"
     8          elif test -d "$i"; then
     9                  echo "d - $i"
    10          else
    11                  echo "u - $i"
    12          fi
    13  done
    14 echo "total $n"

razvan@anaconda:~/uso/scripting$ bash for_ls.bash
f - do_until10.bash
f - echo.bash
f - for10.bash
f - for10_exp.bash
f - for10_seq.bash
f - for_ls.bash
f - here.bash
f - hw.bash
d - test_dir
f - while10.bash
```

```
28 total 10
```

Folosirea caracterului `*` (linia 4) înseamnă expandarea conținutului directorului curent într-o listă de intrări care este parcursă cu ajutorul variabilei `i`. Variabila `n` (linia 3) este folosită pentru contorizarea numărului de intrări (linia 5) folosind expandare aritmetică. Se recomandă folosirea ghilimelelor în lucrul cu fișierele (linia 6) pentru a preveni apariția caracterelor cu rol special în shell (vezi secțiunea 12.4.2).

Instrucțiunea while

Calculul sumei primelor 10 numere naturale folosind `while` se realizează cu scriptul de mai jos:

```
1  razvan@anaconda:~/uso/scripting$ cat -n while10.bash
2       1  #!/bin/bash
3       2
4       3  i=1
5       4  n=10
6       5  sum=0
7       6
8       7  while test $i -le $n; do
9       8          sum=$(($sum + $i))
10      9          ((i++))
11     10  done
12     11
13     12  echo "Suma este: $sum"
14
15 razvan@anaconda:~/uso/scripting$ bash while10.bash
16 Suma este: 55
```

Sintaxa `while` este

```
1 while conditie; do
2   actiuni
3 done
```

`conditie` urmează aceleași reguli ca în cazul `if`. Exemplul prezentat seamănă foarte mult cu un program C: inițializări (liniile 2, 3, 4), condiție (linia 7), acțiuni (linia 8), incrementare (linia 9). Se poate observa că incrementarea lui `i` (linia 9) s-a realizat prin intermediul operatorului de expandare aritmetică.

Comanda `while` este folosită cu precădere în momentul în care condiția este dată de valoarea de retur a unei alte comenzi (de multe ori `read`). O emulare a comenzii **`cat`** cu opțiunea `-n` este exemplificată în scriptul următor:

```
1  razvan@anaconda:~/uso/scripting$ cat -n while_cat.bash
2       1  #!/bin/bash
3       2
4       3  i=1
5       4  cat for10.bash | while read a; do
6       5          echo -e "$i:\t$a"
7       6          i=$(($i + 1))
8       7  done
9       8
10
11 razvan@anaconda:~/uso/scripting$ bash while_cat.bash
12 1:       #!/bin/bash
```

```
2:
3:      sum=0
4:      for i in 1 2 3 4 5 6 7 8 9 10; do
5:      sum=$(($sum + $i))
6:      done
7:
8:      echo "Suma este: $sum"
```

Se observă că se citește în variabila `a` câte o linie din fișiereul `for10.bash` până la sfârșitul fișierului.

Instrucțiunea until

Comanda `until` este asemănătoare cu `while`, doar că în acest caz condiția nu este cea de continuare a acțiunilor, ci este cea de oprire. Datorită similarității nu vom insista; scriptul shell care calculează suma primelor 10 numere naturale este prezentat în continuare:

```
razvan@anaconda:~/uso/scripting$ cat -n until10.bash
     1  #!/bin/bash
     2
     3  i=1
     4  sum=0
     5  n=10
     6
     7  until test $i -gt $n; do
     8          sum=$(($sum + $i))
     9          i=$(($i + 1))
    10  done
    11
    12  echo "Suma este: $sum"

razvan@anaconda:~/uso/scripting$ bash until10.bash
Suma este: 55
```

Exemplele prezentate în secțiunea 12.4.2 sunt pur academice. Nu există niciun motiv întemeiat pentru a calcula suma numerelor naturale sau a emula comenzi existente folosind un script shell. Rolul unui script shell este acela de a folosi cele mai potrivite utilitare existente pentru a rezolva o problemă.

12.5 Filtre de text

Utilitarele de filtrare a textului primesc la intrare un fișier text și oferă la ieșire o formă prelucrată a acestuia. Prelucrarea poate impune schimbarea ordinii elementelor, selecția anumitor elemente (linii, coloane), substituția unor elemente (linii, cuvinte, caractere), adăugarea de noi elemente (număr de linie etc.) și alte operații. Aceste utilitare sunt, de obicei, folosite în tandem cu instrucțiuni de decizie sau de cicluri shell pentru obținerea unor informații ce pot fi utilizate ulterior.

12.5.1 cat, tac, nl

Utilitarul principal în prelucrarea de șiruri este **cat**. **cat** este folosit pentru afișarea unui fișier și (de cele mai multe ori) este prima comandă dintr-o înlănțuire de comenzi care folosesc `|`. După cum s-a precizat în secțiunile anterioare, o opțiune utilă a **cat** este `-n` care afișează și numărul liniei:

```
razvan@anaconda:~/uso/scripting$ cat -n nume.txt
     1  sorin
     2  stefania
     3  mihaela
     4  florin
     5  codrin
     6  lucian
     7  razvan
     8  tavi
```

Utilitarul **tac** (**cat** inversat) este folosit pentru afișarea unui fișier cu liniile inversate (prima linie ultima):

```
razvan@anaconda:~/uso/scripting$ tac nume.txt
tavi
razvan
lucian
codrin
florin
mihaela
stefania
sorin
```

Utilitarul **nl** este folosit pentru a afișa liniile împreună cu numărul lor:

```
razvan@anaconda:~/uso/scripting$ nl for10_exp.bash
     1  #!/bin/bash

     2  sum=0
     3  for ((i = 1; i <= 100; i++)); do
     4          sum=$(($sum + $i))
     5  done

     6  echo "Suma este: $sum"
```

Deosebirea față de **cat -n** este faptul că **nl** nu consideră liniile goale. De asemenea, **nl** are opțiuni pentru paginare, considerare de secțiuni etc. care depășesc aria de cuprindere a acestei cărți.

12.5.2 sort, uniq

Comanda **sort** este folosită pentru sortarea liniilor primite la intrare. În exemplul de mai jos, se realizează sortarea liniilor din fișierul `nume.txt`:

```
razvan@anaconda:~/uso/scripting$ cat nume.txt
sorin
stefania
mihaela
florin
```

```
codrin
lucian
razvan
tavi

razvan@anaconda:~/uso/scripting$ sort nume.txt
codrin
florin
lucian
mihaela
razvan
sorin
stefania
tavi
```

Se poate întâmpla ca un nume să apară de două ori și să avem nevoie doar de o singură apariție în ieșire. În această situație trebuie folosit utilitarul **`uniq`** sau opțiunea `-u` la sort:

```
razvan@anaconda:~/uso/scripting$ sort nume2.txt
alex
alina
alina
cristi
cristi
cristi
razvan
razvan

razvan@anaconda:~/uso/scripting$ sort nume2.txt | uniq
alex
alina
cristi
razvan

razvan@anaconda:~/uso/scripting$ sort -u nume2.txt
alex
alina
cristi
razvan
```

Un exemplu util a fost prezentat în secțiunea 12.4.2. În acel exemplu se dorea aflarea numărului de utilizatori care s-au autentificat în sistem într-o zi dată. Comanda utilizată a fost:

```
razvan@anaconda:~/uso/scripting$ last -30 | grep Mon | cut -d' ' -f1 |
sort | uniq | wc -l
9
```

Opțiuni utile `sort` Dintre opțiunile utile ale comenzii sort amintim:

- `-u`: elimină duplicatele după sortare;
- `-r`: sortare inversă (în ordine descrescătoare);
- `-t`: specificarea separatorului folosit pentru sortare; implicit, sortarea se realizează după primul câmp;
- `-n`: sortare numerică; implicit sortarea este alfanumerică, după cum se poate vedea în exemplul de mai jos:

```
razvan@anaconda:~/uso/scripting$ sort numere.txt
-3
10
100
12
15
2101
28
492
5

razvan@anaconda:~/uso/scripting$ sort -n numere.txt
-3
5
10
12
15
28
100
492
2101
```

Sortare avansată Uneori poate apărea situația în care dorim sortarea după o coloană a fișierului de intrare. Presupunem că avem o miniagendă cu persoane în care intrările sunt în forma `prenume, nume, număr de telefon, adresă de e-mail` și dorim sortarea după nume (adică după a doua coloană). Pentru aceasta vom folosi comanda:

```
razvan@anaconda:~/uso/scripting$ cat nume3.txt
Adrian,Munteanu,0711 111 111,am@ex.com
Ilinca,Zafiu,0711 111 112,iz@ex.com
Andreea,Popa,0711 111 113,ap@ex.com
Dan,Badea,0711 111 114,db@ex.com

razvan@anaconda:~/uso/scripting$ sort -t , -k 2,2 nume3.txt
Dan,Badea,0711 111 114,db@ex.com
Adrian,Munteanu,0711 111 111,am@ex.com
Andreea,Popa,0711 111 113,ap@ex.com
Ilinca,Zafiu,0711 111 112,iz@ex.com
```

Am folosit argumentul `-t` pentru a specifica virgula ca separator. Pentru sortarea după o cheie specifică se folosește argumentul `-k`. Acesta specifică de la ce câmp se începe sortarea (aici este vorba de al doilea câmp) și cu ce câmp se termină (în cazul de față tot 2: `-k 2,2`). În caz de egalitate, se poate alege un nou câmp de sortare prin specificarea unui nou argument `-k`.

12.5.3 head, tail

Comenzile **`head`** și **`tail`** sunt utile pentru a reține primele sau ultimele linii dintr-un fișier primit la intrare. Implicit se reține primele sau ultimele 10 linii. Exemple de utilizare sunt prezentate în continuare:

```
razvan@anaconda:~/uso/scripting$ head /etc/passwd
```

```
razvan@anaconda:~/uso/scripting$ tail /etc/passwd

razvan@anaconda:~/uso/scripting$ head -n 20 /etc/passwd

razvan@anaconda:~/uso/scripting$ tail -n 20 /etc/passwd
```

Comenzile specificate afișează respectiv: primele 10 linii din fișierul `/etc/passwd`, ultimele 10 linii, primele 20 de linii, ultimele 20 de linii.

Comanda **`tail`** este utilă pentru inspecția jurnalelor (log-urilor). Jurnalele sunt de obicei fișiere de mari dimensiuni și, pentru acestea, de cele mai multe ori este suficientă vizualizarea ultimelor intrări.

O opțiune utilă a comenzii **`tail`** este `-f`. Aceasta permite vizualizarea conținutului unui fișier în timp real în sensul că așteaptă scrierea unor noi linii în fișier și afișarea acestora pe măsură ce acestea sunt adăugate:

```
root@anaconda:~# tail -f /var/log/apache/access.log
```

Întreruperea rulării comenzii se încheie prin apăsarea `CTRL-C` (transmiterea semnalului `SIGINT`).

12.5.4 cut

Comanda **`cut`** este utilă pentru a extrage coloane dintr-un fișier format corespunzător. Spre exemplu, dacă folosim fișierul prezentat în secțiunea anterioară și vrem afișarea prenumelui și a adresei de e-mail vom folosi comanda:

```
razvan@anaconda:~/uso/scripting$ cut -d ',' -f 1,4 < nume3.txt
Adrian,am@ex.com
Ilinca,iz@ex.com
Andreea,ap@ex.com
Dan,db@ex.com
```

Cele două opțiuni cele mai importante ale comenzii cut sunt **`-d`** pentru a specifica delimitatorul și **`-f`** pentru a preciza câmpurile (coloanele) care se doresc extrase. În exemplul de mai sus, separatorul a fost virgulă și s-au reținut câmpurile (coloanele) 1 și 4 corespunzătoare prenumelui și adresei de e-mail.

Dacă se dorește extragerea numelui de utilizator și al directorului de bază pentru utilizatorii care au directorul de bază în `/home`, vom folosi următoarea comandă:

```
razvan@anaconda:~/uso/scripting$ cat /etc/passwd | grep /home | cut -d
':' -f1,6
ftp:/home/ftp
george:/home/students/george
andreic:/home/students/andreic
iepurasu:/home/students/iepurasu
valentin:/home/students/valentin
dhartescu:/home/students/dhartescu
ciconaru:/home/students/ciconaru
```

Comanda de mai sus extrage utilizatorii care au directorul de bază în `/home` (**`cat /etc/passwd | grep /home`**), după care extrage numele de utilizator și directorul de bază asociat (**`cut -d ,:' -f 1,6`**). Separatorul de câmp în `/etc/passwd`

este `:`, iar indexurile de câmp corespunzătoare numelui de utilizator și directorului de bază sunt 1 și 6.

12.5.5 tr

Utilitarul **`tr`** (*transliterate*) este folosit pentru translatarea la nivel de caracter a informațiilor de la intrare. Acțiuni posibile care pot fi efectuate cu ajutorul comenzii **`tr`** sunt:

- translatarea caracterelor,
- eliminarea caracterelor care se repetă (*squeeze*),
- ștergerea de caractere.

Exemple de translatarea caracterelor sunt:

```
1  razvan@anaconda:~/uso/scripting$ tr a b <<<"acadaeaf"
2  bcbdbebf
3
4  razvan@anaconda:~/uso/scripting$ tr acd xyz <<<"acadaeaf"
5  xyxzxexf
6
7  razvan@anaconda:~/uso/scripting$ tr a-z A-Z <<<"acadaeaf"
8  ACADAEAF
9
10 razvan@anaconda:~/uso/scripting$ tr '[:lower:]' '[:upper:]' <<<"acadaeaf"
11 ACADAEAF
```

În primul exemplu se translatează caracterul `a` în caracterul `b`. Şirul este transmis sub formă de *here string*. În cel de-al doilea exemplu, avem un set de intrare (`acd`) și un set de ieșire (`xyz`). Translatarea se face între primul element din setul de intrare și primul element din setul de ieșire, al doilea element din setul de intrare și al doilea element din setul de ieșire etc. Al treilea și al patrulea exemplu translatează literele mari în litere mici. `a-z` simbolizează literele de la `a` la `z`. De asemenea, expresiile `[:lower:]` și `[:upper:]` reprezintă, respectiv, literele mici și literele mari din alfabet. Utilitarul **`tr`** folosește mai multe astfel de expresii. Mai multe detalii puteți afla din pagina de manual sau info (**`man tr`**, **`info coreutils`**).

Exemple de eliminare a caracterelor care se repetă sunt următoarele:

```
1  razvan@anaconda:~/uso/scripting$ tr -s ' ' <<< "a    b    c"
2  a b c
3
4  razvan@anaconda:~/uso/scripting$ tr -s a-z A-Z <<< "aa    bb    cc"
5  A    B    C
```

În cazul primului exemplu, caracterul spațiu este eliminat până la o singură apariție. Opțiunea `-s` este folosită pentru a specifica acțiunea de eliminare a duplicatelor unui caracter (*squeeze*).

În cel de-al doilea exemplu se folosește simultan operațiunea de translatare și de eliminare de duplicate pe șirul de intrare "aa bb cc". Rezultatul este transformarea literelor mici în litere mari simultan cu eliminarea duplicatelor acestora.

Ştergerea caracterelor se face cu ajutorul opțiunii `-d`:

```
razvan@anaconda:~/uso/scripting$ tr -d 'a' <<< 'abcdcba'
bcdcb

razvan@anaconda:~/uso/scripting$ tr -ds 'a' 'bcd' <<< 'aabbccddccbbaa'
bcdcb
```

Opțiunea `-d` primește un set de caractere care sunt eliminate. Opțiunea `-d` poate fi completată de opțiunea `-s`. În cel de-al doilea exemplu se șterge caracterul `a` din șirul de intrare iar celorlalte caractere le sunt eliminate duplicatele.

12.5.6 wc

Utilitarul **`wc`** (*word count*) permite contorizarea numărului de linii, de caractere sau de cuvinte dintr-un fișier text. La o folosire fără opțiuni, **`wc`** afișează atât numărul de linii cât și numărul de cuvinte și caractere:

```
razvan@anaconda:~/uso/scripting$ wc < for10.bash
  8  24 102
```

Pentru afișarea numai a numărului de linii, de cuvinte sau de caractere, se folosesc, respectiv, opțiunile `-l`, `-w`, `-c`:

```
razvan@anaconda:~/uso/scripting$ wc -l < for10.bash
8

razvan@anaconda:~/uso/scripting$ wc -w < for10.bash
24

razvan@anaconda:~/uso/scripting$ wc -c < for10.bash
102
```

12.5.7 grep

Utilitarul **`grep`** și variantele sale (**`egrep`**, **`fgrep`**) sunt utilitarele de bază în prelucrarea fișierelor text. Acest utilitar permite selectarea anumitor linii dintr-un fișier text pe baza unei *expresii regulate* transmise ca argument. **`grep`** primește ca parametru șirul de căutare și, eventual, fișierul în care se face căutarea. Dacă dorim să vedem informații despre autentificarea utilizatorului `adrian` în sistem vom folosi comanda:

```
razvan@anaconda:~/uso/scripting$ last | grep adrian
adrian   pts/2        92.80.182.158    Sun Sep  2 22:30 - 22:38  (00:07)
adrian   pts/2        92.80.149.10     Sun Sep  2 09:10 - 09:29  (00:18)
adrian   pts/0        92.80.149.10     Sun Sep  2 09:06 - 09:29  (00:23)
```

Dacă dorim să afișăm informații despre utilizatorii din sistem al căror nume începe cu litera `a` vom folosi comanda:

```
razvan@anaconda:~/uso/scripting$ cat /etc/passwd | grep '^a'
alexpoke:x:1003:1003:Poke Alexandu,,,,A & C:/home/alexpoke:/bin/bash
adrian:x:1009:1009:Adrian Nistor,,,:/home/adrian:/bin/bash
amihaiuc:x:1011:1011:Alex Mihaiuc,,,:/home/amihaiuc:/bin/bash
alina:x:1016:1016:Alina Deaconescu,,,:/home/alina:/bin/bash
andrewbwm:x:1031:1026:Andrei Buhaiu:/home/students/andrewbwm:/bin/bash
```

```
alexef:x:1032:1026:Alex Eftimie:/home/students/alexef:/bin/bash
andreic:x:1045:1026:Andrei Cibotaru:/home/students/andreic:/bin/bash
```

În cazul de față, pentru obținerea utilizatorilor al căror nume începe cu litera `a` se folosește expresia regulată `^a`. Utilizarea ei va duce la selectarea acelor linii care încep cu litera `a` (caracterul `^` este special și simbolizează început de linie). Mai multe detalii despre *expresii regulate* **`grep`** se găsesc mai jos:

`grep` este utilizabil cu foarte multe opțiuni. O parte din cele mai importante sunt exemplificate în continuare pentru fișierul `grep_test.txt`:

```
razvan@anaconda:~/uso/scripting$ cat grep_test.txt
alfa beta
AlFa BeTa
Beta gamma
beta delta
deltadd epsilon
```

- opțiunea `-i` ignoră literele mari (*ignore case*):

```
razvan@anaconda:~/uso/scripting$ grep -i "alfa" < grep_test.txt
alfa beta
AlFa BeTa
```

- opțiunea `-v` inversează căutarea (afișează liniile care nu conțin expresia de căutare):

```
razvan@anaconda:~/uso/scripting$ grep -v "alfa" < grep_test.txt
AlFa BeTa
Beta gamma
beta delta
deltadd epsilon

razvan@anaconda:~/uso/scripting$ grep -v -i "alfa" < grep_test.txt
Beta gamma
beta delta
deltadd epsilon
```

- opțiunea `-n` afișează numărul liniei:

```
razvan@anaconda:~/uso/scripting$ grep -n "alfa" < grep_test.txt
1:alfa beta
```

- opțiunea `-w` este folosită pentru căutarea unui cuvânt întreg:

```
razvan@anaconda:~/uso/scripting$ grep -w "delta" < grep_test.txt
beta delta
```

- opțiunea `-r` face căutare recursivă în structura de directoare:

```
razvan@anaconda:~/uso/scripting$ grep -r "for" .

./for_ls.bash:for i in *; do
./while_cat.bash:cat for10.bash | while read a; do
./for10.bash:for i in 1 2 3 4 5 6 7 8 9 10; do
./for10_seq.bash:for i in $(seq 1 100); do
./for10_exp.bash:for ((i = 1; i <= 100; i++)); do
./if_user.bash: echo "Bow before me for I am root."
```

Rezultatul dat de utilizarea acestei opțiuni prezintă fișierele în care s-a găsit acel șir/expresie regulată și linia completă/liniile complete din fișiere.

Expresii regulate grep

După cum s-a observat, **`grep`** folosește un șir de căutare care poate avea forma unei expresii regulate.

> O **expresie regulată** este un șir de caractere utilizat pentru a se potrivi cu un alte șiruri de caractere conform unor reguli de sintaxa bine precizate. O expresie regulată conține caractere simple și caractere cu semnificație specială.

O parte din caracterele/grupările de caractere cu semnificație specială sunt prezentate în continuare:

- Caracterele `[]` sunt folosite pentru a defini *un set de caractere* de potrivit; exemple:
  - `[abcd]` înseamnă potrivire cu oricare din caracterele `a`, `b`, `c`, `d`;
  - `[a-z]` înseamnă potrivire cu oricare minusculă;
  - `[0-9a-zA-Z]` înseamnă potrivire cu orice caracter alfanumeric;
  - `[^A-Z]` înseamnă potrivire cu orice caracter mai puțin majusculele.
- Caracterul `.` înseamnă orice caracter;
- Caracterul `^` poziționat în afara unui set de caractere înseamnă potrivire cu începutul liniei;
- Caracterul `$` înseamnă potrivire cu sfârșitul liniei;
- Simbolurile `\<`, respectiv `\>` sunt folosite pentru a se potrivi cu șirul vid de la început și sfârșit de cuvânt;
- Simbolul `?` înseamnă potrivire de cel mult o dată a caracterului/grupului de caractere anterioare;
- Simbolul `*` înseamnă potrivire de zero sau mai multe ori a caracterului/grupului de caractere anterioare;
- Simbolul `+` înseamnă potrivire de o dată sau mai multe ori a caracterului/grupului de caractere anterioare;
- `{n}` înseamnă potrivire de `n` ori;
- `{n,}` înseamnă potrivire de cel puțin `n` ori;
- `{n,m}` înseamnă potrivire de cel puțin `n` ori și cel mult `m` ori.

Caracterele/grupurile de caractere prezentate mai sus pot fi concatenate pentru generarea unei expresii regulate complexe.

Informații complete despre comenzile de filtrare de text folosite în shell scripting găsiți în paginile info asociate (**`info coreutils`**).

12.5.8 sed

Utilitarul **sed** (stream editor) este folosit pentru a aplica transformări textuale unui flux de date. În mod implicit, citește de la intrare o linie, aplică operația specificată, după care afișează linia modificată.

sed poate fi considerat un editor non-interactiv. Acesta aplică un set de operații fiecărei linii din fișierul de la intrare. Întrucât în memorie se reține o singură linie, prelucrarea se poate face pe un fișier cu un număr arbitrar de linii.

Pentru a exemplifica modul de utilizare a **sed** vom folosi următorul fișier de lucru:

```
1 razvan@anaconda:~/uso/scripting$ cat sed_ex.txt
2 Andreea Popescu
3 Alin Ionescu
4 Mihai Francu
5 Calin Antonescu
6 Silvia Asavei
7 Doina Ignat
```

Sintaxa de rulare

Un exemplu obișnuit de rulare a **sed** este înlocuirea unui text cu un altul:

```
1 razvan@anaconda:~/uso/scripting$ sed 's/Alin/ALIN/' sed_ex.txt
2 Andreea Popescu
3 ALIN Ionescu
4 [...]
```

După cum se observă, sintaxa de rulare **sed** este:

```
1 sed comenzi fisier_intrare
```

În cazul în care fișierul de intrare lipsește, se folosește intrare standard. De cele mai multe ori, comenzile sunt date între caractere apostrof pentru a evita folosirea caracterelor speciale shell.

În mod implicit, **sed** afișează liniile de la intrare la ieșire. Pentru a dezactiva acest lucru se folosește opțiunea `-n`.

Sintaxa comenzi sed

După cum s-a putut observa și la exemplele anterioare, o comandă **sed** are sintaxa `[adresă][functie][argumente]`.

Adresa definește spațiul din fișier la care se aplică funcția. *Adresa* poate fi o linie a fișierului, un spațiu de linii sau o expresie ce identifică linia. În acest fel:

- `1 s/a/A/` – înlocuiește litera `a` cu `A` în prima linie a fișierului;
- `1,10 s/a/A/` – înlocuiește litera `a` cu `A` în primele zece linii ale fișierului;
- `100,$ s/a/A/` – înlocuiește litera `a` cu `A` de la linia 100 până la sfârșitul fișierului;
- `/alfa/ s/a/A/` – înlocuiește litera `a` cu `A` numai în liniile în care apare șirul `alfa`.

Funcții sed

Funcțiile cele mai utile **sed** sunt prezentate în continuare.

Funcția de substituție (s) a lui **sed** este `s` și a fost folosită în exemplele anterioare. Aceasta folosește ca argumente șirul de înlocuit și cel înlocuitor. Separatorul folosit implicit este `/` (*slash*), dar se poate folosi oricare altul:

```
s/sir_de_inlocuit/sir_inlocuitor/optiuni
```

Opțiuni utile ale funcției de substituție sunt:

- `g` (*global*) – înlocuiește toate aparițiile șirului de înlocuit
- `p` (*print*) – afișează linia la ieșire; folosit, de obicei, în conjuncție cu opțiunea `-n` pentru **sed**
- `w` (*write*) – scrie linia într-un fișier primit ca parametru

Funcția quit (q) oprește citirea de la intrare a fișierului. Exemple de folosire sunt afișarea primelor `N` linii din fișierul de la intrare:

```
razvan@anaconda:~/uso/scripting$ sed '2q' sed_ex.txt
Andreea Popescu
Alin Ionescu
```

sau afișarea primelor linii până la găsirea șirului `Calin`:

```
razvan@anaconda:~/uso/scripting$ sed '/Calin/q' sed_ex.txt
Andreea Popescu
Alin Ionescu
Mihai Francu
Calin Antonescu
```

Funcțiile print și delete (p și d) afișează, respectiv șterg linia de la intrare la ieșire.

Expresii regulate în sed

Funcțiile **sed** pot avea argumente. Întrucât **sed** lucrează pe intrare în format text, argumentele pot fi exprimate ca expresii regulate. Acestea sunt similare cu expresiile regulate grep (vezi secțiunea 12.5.7). Mai multe detalii despre acestea puteți găsi în pagina info **sed**, folosind comanda:

```
info sed "Regular Expressions"
```

Exemple

Mai jos se pot regăsi câteva exemple de utilizare a comenzii **sed**:

- translatarea unui fișier în format DOS (CR/LF) în format Unix:

```
sed 's/.$//'
```

Se observă că se elimină ultimul caracter de la sfârșitul liniei (în cazul de față caracterul `\r` – *Carriage Return*)

- translatarea unui fișier în format Unix în format DOS:

```
1 sed 's/$/\r/'
```

  Se adaugă la sfârșitul liniei caracterul `\r`.

- afișarea tuturor liniilor mai puțin ultimele 100:

```
1 sed '1,2800d'
```

- ștergerea spațiilor albe de la începutul liniilor unui fișier:

```
1 sed 's/[ \t]*//'
```

- ștergerea spațiilor albe de la sfârșitul liniilor de fișier:

```
1 sed 's/[ \t]*$//'
```

- ștergerea liniilor goale:

```
1 sed 's/^[ \t]*$//'
```

- afișarea liniilor care conțin un șir de căutare (emulare **grep**):

```
1 sed -n '/regex/p'
```

- afișarea într-un fișier a liniilor care conțin un anumit șir:

```
1 sed '/regex/w out.txt'
```

- substituția de coloane; în fișierul de exemplu vrem să înlocuim numele cu prenumele:

```
1 razvan@anaconda:~/uso/scripting$ sed 's/\([a-zA-Z]*\) \([a-zA-Z]*\)
  /\2 \1/' sed_ex.txt
2 Popescu Andreea
3 Ionescu Alin
4 Francu Mihai
5 Antonescu Calin
6 Asavei Silvia
7 Ignat Doina
```

  În exemplul de mai sus se folosește construcția `\(regex\)` pentru a stoca valoarea expresiei regulate. Aceste valori sunt folosite, respectiv, cu ajutorul construcțiilor `\1`, `\2`, `\3` etc. În situația de mai sus, numele este reținut în construcția `\2` iar prenumele în `\1`. Afișarea se face în formatul nume prenume.

12.5.9 awk

awk este un utilitar și un limbaj de programare folosit pentru prelucrarea textului pe sisteme Unix. Există mai multe versiuni de awk, versiunea cea mai întâlnită pe sisteme Linux fiind implementarea GNU, **gawk**[1]. Limbajul awk are o sintaxă asemănătoare cu limbajul C care permite operații avasate pentru formatarea fișierului text. Denumirea awk provine de la inițialele celor care au proiectat limbajul: Alfred V. Aho, Peter J. Weinberger și Brian W. Kernighan.

[1] http://www.gnu.org/software/gawk/

În forma sa cea mai simplă, awk poate fi folosit ca o versiune avansată a utilitarului **cut**. Dacă, spre exemplu, se dorește doar extragerea numelui de utilizator și a directorului home folosind fișierul `/etc/passwd` următoarele două comenzi sunt echivalente:

```
razvan@valhalla:~/carte-uso.git$ cut -d ':' -f 1,6 /etc/passwd

razvan@valhalla:~/carte-uso.git$ awk -F ':' '{ print $1,":",$6;}' /etc/
passwd
```

Comenzile de mai sus reține primul și al șaselea câmp al fiecărei linii din fișierul `/etc/passwd` folosind separatorul `:`. Opțiunea `-F` a awk precizează separatorul folosit. Argumentele `$1`, respectiv `$6` sunt folosite pentru extrage primul și al șaselea câmp.

Un avantaj important al **awk** față de **cut** este posibilitatea formatării ieșirii. În vreme ce **cut** selectează anumite coloane, **awk** permite și prelucrarea ieșirii selectate. Astfel, dacă dorim ca informații afișate să fie încadrate de caracterul `|` (*pipe*), se va folosi următoarea comandă:

```
razvan@valhalla:~/carte-uso.git$ awk -F ':' '{ printf "|%-15s|%-25s|\n",
$1, $6;}' /etc/passwd
|root           |/root                     |
|daemon         |/usr/sbin                 |
|bin            |/bin                      |
|sys            |/dev                      |
|sync           |/bin                      |
|games          |/usr/games                |
|man            |/var/cache/man            |
[...]
```

Instrucțiunea `printf` folosită în exemplul de mai sus are o sintaxă similară celei din C, cu absența parantezelor.

Sintaxa awk este similară sintaxei sed:

```
awk [options] ' program instructions ' [input file]
```

În general, programul awk este aplicat fiecărei linii citită de la intrare. awk permite folosirea clauzelor speciale `BEGIN` și `END` care permit aplicarea instrucțiunilor asociate doar la începutul și sfârșitul prelucrării. Astfel, pentru a încadra ieșirea de mai sus, se folosește o comandă ca cea de mai jos. Programul se poate introduce pe mai multe linii, până la "închiderea" acestuia folosind caracterul `'` (apostrof):

```
razvan@valhalla:~/carte-uso.git$ awk -F ':' '
> BEGIN { printf "+---------------+-------------------------+\n"; }
>       { printf "|%-15s|%-25s|\n", $1, $6; }
> END   { printf "+---------------+-------------------------+\n"; }
> ' /etc/passwd
+---------------+-------------------------+
|root           |/root                     |
|daemon         |/usr/sbin                 |
|bin            |/bin                      |
|sys            |/dev                      |
[...]
|pulse          |/var/run/pulse            |
|saned          |/home/saned               |
+---------------+-------------------------+
```

awk folosește instrucțiuni și comenzi similare limbajului C. Dacă se dorește afișarea din 5 în 5 a utilizatorilor sistemului și a directorului home asociat se poate folosi o comandă de forma:

```
razvan@valhalla:~/school/2009-2010/uso/carte-uso.git$ awk -F ':' -f
print_users.awk /etc/passwd
+---------------+--------------------------+
|root           |/root                     |
|games          |/usr/games                |
|uucp           |/var/spool/uucp           |
|irc            |/var/run/ircd             |
|messagebus     |/var/run/dbus             |
|gdm            |/var/lib/gdm              |
|mpd            |/var/lib/mpd              |
|postfix        |/var/spool/postfix        |
|pulse          |/var/run/pulse            |
+---------------+--------------------------+
```

Opțiunea `-f` permite selectarea fișierul care să conțină programul awk, prezentat în listingul 12.5.9. Variabila `current_users` reține numărul de utilizatori curent. Este inițializată cu 0 la începutul prelucrării (în cadrul blocului `BEGIN`) și apoi este incrementată pe parcursul prelucrării liniilor. Se folosește instrucțiunea `if` pentru a selecta doar anumiți utilizatori.

```
#!/usr/bin/awk -f

BEGIN {
        current_users = 0;
        printf "+---------------+--------------------------+\n";
}
{
        current_users++;
        if (current_users % 5 == 1)
                printf "|%-15s|%-25s|\n", $1, $6;
}
END {
        printf "+---------------+--------------------------+\n";
}
```

Listing 12.1: Afișare utilizatorilor din 5 în 5

awk permite folosirea expresiilor regulate pentru selectarea doar anumitor linii. Astfel, dacă se dorește afișarea doar acelor utilizatori care au directorul home în `/home` se folosește un program ca cel prezentat în listingul 12.2. Expresiile regulate se scriu între caractere / (slash) similar utilitarului `sed`. Rezultatul rulării acestui program este:

```
razvan@valhalla:~/carte-uso.git$ awk -F ':' -f print-home-users.awk /etc/
passwd
+---------------+--------------------------+
|razvan         |/home/razvan              |
|festival       |/home/festival            |
|ntp            |/home/ntp                 |
|alina          |/home/alina               |
|ftp            |/home/ftp                 |
|saned          |/home/saned               |
+---------------+--------------------------+
```

```
#!/usr/bin/awk -f

```

```
3  BEGIN {
4          printf "+---------------+-------------------------+\n";
5  }
6  /\/home/ {
7          printf "|%-15s|%-25s|\n", $1, $6;
8  }
9  END {
10         printf "+---------------+-------------------------+\n";
11 }
```

Listing 12.2: Afișare utilizatori /home

În listingul 12.2 se verifică dacă fiecare linie conține șirul `/home`. De fapt, funcționarea corectă a programului se bazează doar pe verificarea faptului că a șasea coloană conține șirul `/home`. Acest lucru se poate realiza ușor în awk, după cum este prezentat în listingul 12.8.1. Construcția `$6 ~ /^\/home/` verifică dacă al șaselea câmp al unei linii începe cu șirul `/home`.

```
1  #!/usr/bin/awk -f
2
3  BEGIN {
4          printf "+---------------+-------------------------+\n";
5  }
6  $6 ~ /^\/home/ {
7          printf "|%-15s|%-25s|\n", $1, $6;
8  }
9  END {
10         printf "+---------------+-------------------------+\n";
11 }
```

Listing 12.3: Afișare utilizatori /home folosind expresii regulate

Un alt avantaj important al comenzii `awk` față de comanda `cut` este folosirea unei expresii regulate pe post de separator. Opțiunea `-F` a awk primește ca argument o expresie regulată care este folosită pentru a separa câmpurile unei linii. Următoarele două comenzi, pentru determinarea adresei hardware a interfeței eth0 sunt echivalente:

```
1  razvan@valhalla:~/carte-uso.git$ /sbin/ifconfig eth0 | head -1 | tr -s '
   ' | cut -d ' ' -f 5
2  00:1d:09:b4:0c:26
3
4  razvan@valhalla:~/carte-uso.git$ /sbin/ifconfig eth0 | head -1 | awk -F
   '[
5  \t]+' '{ print \$5; }'
6  00:1d:09:b4:0c:26
```

Expresia regulată `[ \t]+` specifică faptul că orice set de spații albe (caractere blank sau TAB) va fi folosit ca separator.

awk oferă un set mult mai amplu de funcționalități. Mai multe detalii despre acestea și despre cazuri posibile de utilizare găsiți în manualul Gawk[1].

[1] http://www.gnu.org/software/gawk/manual/

12.6 Comenzi de lucru cu fișiere

Comenzile de lucru cu fișiere sunt comenzi care permit căutarea în sistemul de fișiere și executarea unor acțiuni pentru fișierele căutate. Comenzile prezentate în această secțiune sunt **xargs**, **locate** și **find**.

12.6.1 xargs

Comanda **xargs** permite transmiterea de argumente unei alte comenzi, argumente ce pot fi trimise de la intrarea standard. Spre exemplu, dacă avem un fișier ce conține o listă fișiere pe care dorim să le ștergem (câte unul pe linie), vom folosi comanda: **xargs rm < file.txt** ca în exemplul de mai jos

```
1  razvan@anaconda:~/uso/scripting$ cat file.txt
2  tmp1.txt
3  tmp2.txt
4  tmp3.txt
5
6  razvan@anaconda:~/uso/scripting$ ls tmp*
7  tmp1.txt  tmp2.txt  tmp3.txt
8  razvan@anaconda:~/uso/scripting$ xargs rm < file.txt
9
10 razvan@anaconda:~/uso/scripting$ ls tmp*
11 ls: tmp*: No such file or directory
```

În exemplu, fișierul file conținea trei nume de fișier: `tmp1.txt`, `tmp2.txt`, `tmp3.txt`. Folosind **xargs** se invocă **rm**. Argumentele pentru **rm** sunt citite de la intrarea standard în care a fost redirectat fișierul `file.txt`. Rezultatul este ștergerea fișierelor `tmp1.txt`, `tmp2.txt` și `tmp3.txt` precizate în fișierul file.txt.

Dacă un utilizator dorește afișarea fișierelor ce conțin șirul `for` va utiliza comanda:

```
1  ubuntu@ubuntu:~$ grep -l 'for' *
```

sau

```
1  ubuntu@ubuntu:~$ ls | xargs grep -l 'for'
```

La fel, dacă se dorește generarea unei liste a utilizatorilor în sistem se poate utiliza comanda:

```
1  razvan@anaconda:~/uso/scripting$ cut -d ':' -f 1 < /etc/passwd | sort |
   tr -s '\n' ' '
2  Debian-exim adrian alexef alexpoke alina amihaiuc [...]
```

sau

```
1  razvan@anaconda:~/uso/scripting$ cut -d ':' -f 1 < /etc/passwd | sort |
   xargs echo
2  Debian-exim adrian alexef alexpoke alina amihaiuc [...]
```

Opțiuni utile pentru **xargs** sunt:

- `-d` este folosită pentru a specifica delimitatorul de parametri; implicit acesta este orice caracter alb (*whitespace*); în exemplul de mai jos se șterg fișierele `tmp1.txt`, `tmp2.txt` și `tmp3.txt`:

```
razvan@anaconda:~/uso/scripting$ cat file2.txt
tmp1.txt,tmp2.txt,tmp3.txt

razvan@anaconda:~/uso/scripting$ ls tmp*
tmp1.txt  tmp2.txt  tmp3.txt

razvan@anaconda:~/uso/scripting$ xargs -d ',' rm  < file2.txt
rm: cannot remove '\n': No such file or directory
```

Ultima linie apare ca urmare a faptului că se interpretează caracterul newline (`\n`) ca un argument posibil.

- `-0` impune folosirea ca delimitator de argument a nul-terminatorului de sir (`\0`); acest lucru este util în momentul în care parametrii conțin caractere speciale (blank, backslash etc.); este folosit, de obicei, împreună cu argumentul `-print0` la **`find`**, precum este descris în secțiunea 12.6.3.

- `-I` permite folosirea unui șir care va fi substituit pe parcursul comenzii cu argumentul primit la intrare; în următorul exemplu, fișierele primite la intrare sunt copiate cu extensia `.bkup` în `/tmp`:

```
razvan@anaconda:~/uso/scripting$ cat file3.txt
for10.bash
for10_seq.bash
for10_exp.bash

razvan@anaconda:~/uso/scripting$ xargs -I abc cp abc /tmp/abc.bkup <
 file3.txt

razvan@anaconda:~/uso/scripting$ ls /tmp/*.bkup
/tmp/for10.bash.bkup  /tmp/for10_exp.bash.bkup  /tmp/for10_seq.bash.
bkup
```

Se observă că s-a folosit șirul `abc` ca șir de substituit (prin intermediul opțiunii `-I`). La apariția acestui șir în comanda folosită de **`xargs`**, acesta a fost înlocuit cu parametrul de la intrare (în cazul nostru fiecare din cele trei fișiere). Astfel, comanda

```
ubuntu@ubuntu:~$ xargs -I abc cp abc /tmp/abc.bkup
```

este echivalentă cu

```
ubuntu@ubuntu:~$ cp for10.bash for10.bash.bkup

ubuntu@ubuntu:~$ cp for10.bash for10_seq.bash.bkup

ubuntu@ubuntu:~$ cp for10.bash for10_exp.bash.bkup
```

12.6.2 locate

Comanda **`locate`** permite localizarea de fișiere al căror nume corespunde unui anumit șablon transmis ca parametru. Comanda locate folosește o bază de date construită cu ajutorul comenzii **`updatedb`**.

Un exemplu este prezentat în continuare:

```
razvan@anaconda:~/uso/scripting$ time locate strace
/usr/bin/strace
/usr/share/doc/strace
/usr/share/doc/strace/changelog.Debian.gz
/usr/share/doc/strace/changelog.gz
/usr/share/doc/strace/copyright
/usr/share/doc/strace/TODO.gz
/usr/share/man/man1/strace.1.gz
/usr/share/vim/vim70/syntax/strace.vim
/var/lib/dpkg/info/strace.list

real    0m0.288s
user    0m0.284s
sys     0m0.004s
```

După cum se vede comanda locate este foarte rapidă întrucât folosește baza de date generată/actualizată anterior de **updatedb**. Întrucât este posibil ca anumite fișiere să fi fost șterse de la ultima actualizare, se poate folosi opțiunea `-e` pentru a afișa doar fișierele care există în sistem dintre cele cunoscute:

```
razvan@anaconda:~/uso/scripting$ locate -e strace
/usr/bin/strace
/usr/share/doc/strace
/usr/share/doc/strace/changelog.Debian.gz
/usr/share/doc/strace/changelog.gz
[...]
```

Comanda **locate** este de multe ori folosită împreună cu **xargs**:

```
razvan@anaconda:~/uso/scripting$ locate -e strace | xargs file
/usr/bin/strace:                                ELF 32-bit LSB executable,
Intel 80386, version 1 (SYSV), for GNU/Linux 2.4.1, dynamically linked (
uses shared libs), for GNU/Linux 2.4.1, stripped
/usr/share/doc/strace:                          directory
/usr/share/doc/strace/changelog.Debian.gz: gzip compressed data, was "
changelog.Debian", from Unix, last modified: Fri Oct 27 00:22:12 2006,
max compression
[...]
```

12.6.3 find

Comanda **find** este principala comandă utilizată pentru căutarea în sistemul local de fișiere. Comanda realizează o căutare recursivă în structura de directoare și descoperă acele intrări în sistemul de fișiere care satisfac o anumită condiție. Suplimentar, comanda **find** permite executarea unei acțiuni specifice.

Sintaxa **find** este mai complexă decât a altor comenzi și o vom prezenta după următorul exemplu:

```
razvan@anaconda:~$ find /usr/include/ -type f -name '*term*.h' -print
/usr/include/asm-generic/termios.h
/usr/include/linux/termios.h
/usr/include/asm/termbits.h
/usr/include/asm/termios.h
/usr/include/bits/termios.h
/usr/include/sys/termios.h
```

```
/usr/include/termio.h
/usr/include/termios.h
[...]
```

În exemplul de mai sus, se caută recursiv în `/usr/include` fișierele (`-type f`) al căror nume corespunde expresiei regulate `'*term*.h'` și se afișează acele fișiere (`-print`).

Astfel, sintaxa `find` este

```
find director optiuni_potrivire actiuni
```

unde:

- `director` este directorul în care se face căutarea (recursiv);
- `optiuni_potrivire` sunt opțiuni de căutare în sistemul de fișiere (adâncimea căutării, tipul intrării, nume, deținător, permisiuni etc.);
- `actiuni` reprezintă acțiunile întreprinse în momentul găsirii unei intrării corespunzătoare; acțiunea implicită este -print (de afișare).

Opțiuni de potrivire utile sunt:

- `tipul` intrării se precizează cu ajutorul opțiunii `-type`; astfel, `-type f` înseamnă un fișier, `-type d` înseamnă un director, `-type l` înseamnă o legătură simbolică etc.; exemplu: afișarea legăturilor simbolice din ierarhia utilizatorului curent:

```
razvan@anaconda:~$ find /home/razvan/public_html/ -type l
/home/razvan/public_html/pub/books
/home/razvan/public_html/pictures
/home/razvan/public_html/movies
/home/razvan/public_html/games
/home/razvan/public_html/wiki
/home/razvan/public_html/school/uso/cursuri
[...]
```

- numele intrării se precizează cu ajutorul opțiunii `-name` urmată de o expresie regulată descriind numele; opțiunea `-wholename` este folosită pentru potrivire cu numele complet al intrării (cale absolută); exemplu: afișarea directoarelor ce conțin calea `'c++'`:

```
razvan@anaconda:~$ find /usr/include/ -type d -wholename '*/c++/*'
/usr/include/c++/4.1.2
/usr/include/c++/4.1.2/backward
/usr/include/c++/4.1.2/bits
/usr/include/c++/4.1.2/debug
/usr/include/c++/4.1.2/ext
/usr/include/c++/4.1.2/ext/pb_assoc
[...]
```

- permisiunile se precizează cu ajutorul opțiunii `-perm`; astfel, `-perm 644` va găsi intrările care au drepturi de citire și scriere pentru utilizator și doar de citire pentru grup și pentru ceilalți; exemplu: afișarea executabilelor suidate de sistem:

```
razvan@anaconda:~$ find /bin -perm 4755
/bin/su
/bin/mount
```

```
4 /bin/umount
5 /bin/ping
6 /bin/ping6
```

- timpul ultimului acces sau a ultimei modificări cu ajutorul opțiunilor `-amin`, `-atime`, `-mmin`, `-mtime`; `-amin 20` înseamnă că fișierul a fost accesat în urmă cu 20 de minute, `-amin -20` înseamnă că fișierul a fost accesat în urmă cu cel mult 20 de minute, `-amin +20` înseamnă că fișierul a fost accesat în urmă cu cel puțin 20 de minute; exemplu: aflarea fișierelor modificate în urmă cu mai mult de 100 de zile:

```
1 razvan@anaconda:~$ find /home/razvan/ -type f -mtime +100
```

- spațiul ocupat pe disc cu ajutorul opțiunii `-size`; `-size +10M` va selecta fișierele care ocupă mai mult de 10 MB; exemplu: aflarea fișierelor care ocupă mai mult de 200 MB:

```
1 razvan@anaconda:~$ find /home/razvan/ -size +200M
```

- deținătorul fișierului cu ajutorul opțiunii `-user`;
- grupul fișierului cu ajutorul opțiunii `-group`.

Acțiunile ce pot fi întreprinse cu ajutorul comenzii **`find`** sunt:

- afișarea fișierului se face folosind `-print`; această acțiune este implicită;
- afișarea fișierelor folosind nul-terminatorul ca separator folosind `-print0`; opțiunea este utilă în combinație cu argumentul `-0` la **`xargs`**; exemplu: mutarea fișierelor `.iso` pe o nouă partiție:

```
1 ubuntu@ubuntu:~$ find . -type f -name *.iso -print0 | xargs -0 -I
abc mv abc /mnt/hda6
```

- ștergerea fișierului cu ajutorul `-delete`; exemplu: ștergerea fișierelor temporare din ierarhia utilizatorului local:

```
1 razvan@anaconda:~$ find . -type f -name '*~' -delete
```

- execuția unei comenzi precizate de utilizator cu ajutorul opțiunii `-exec`; șirul `{}` este utilizat pentru înlocuirea intrării găsite în comanda de după `exec`; comanda trebuie să se termine cu `;` (punct și virgulă); se recomandă ca atât șirul `{}` cât și caracterul `;` să fie citate pentru a preveni interpretarea lor ca niște caractere speciale; exemple:
  - crearea unei copii pentru fișierele mai vechi de 1 an din directorul curent:

```
1 razvan@anaconda:~/uso/scripting$ find . -type f -mtime +365 -
exec cp '{}' /tmp/'{}'.bkup ';'
```

  - adăugarea dreptului de execuție pentru toate directoarele:

```
1 razvan@anaconda:~/uso/scripting$ find . -type d -exec chmod a+x
'{}' ';'
```

  - afișarea fișierelor C în care este definită funcția `main`:

```
1 razvan@anaconda:~$ find school/2001-2006_code/ -type f -name '*.
c' -exec grep -l 'int main*' '{}' ';'
```

Acțiuni asupra fișierelor găsite cu find pot fi executate cu ajutorul opțiunii `-exec` sau prin folosirea **`xargs`**.

Exemple

În continuare sunt prezentate câteva exemple utile de lucru cu **`find`**:

- editarea fișierelor `.c` din ierarhia curentă care folosesc funcția `printf`:

```
1 razvan@anaconda:~$ vi $(find school/2001-2006_code/ -type f -name
  '*.c' -exec grep -l -w 'printf' '{}' ';')
```

- arhivarea fișierelor mai vechi de 30 de zile:

```
1 razvan@anaconda:~/uso/scripting$ find . -depth -type f -mtime 30 |
  tar --files-from=- --null -czf out.tgz
```

- ștergerea fișierelor utilizatorului local din `/tmp`:

```
1 razvan@anaconda:~/uso/scripting$ find /tmp -type f -user $(whoami) -
  delete
```


Informații complete de comenzile de lucru cu fișierele se găsesc în paginile info asociate (**`info find`**).

12.7 Expandarea în shell

De-a lungul secțiunilor anterioare s-au prezentat diferite metode de expandare în shell. Expandarea unei variabile la valoarea ei, expandarea unei comenzi, expandare aritmetică. Secțiunea de față oferă o recapitulare structurată a acestor mecanisme.

12.7.1 Simbolul $

Simbolul `$` este folosit pentru expandarea unei variabile, comenzi sau expresii și posibilitatea transmiterii rezultatului acesteia. Caracterul are o semnificație specială, drept pentru care va trebui citat (folosind ghilimele, apostrof, backslash) în cazul în care se dorește folosirea sa literală.

Variabile

O variabilă are un nume și o valoare. Așa cum s-a specificat, o variabilă nu are tip în shell. Poate fi interpretată ca număr sau ca șir în funcție de situație. Spre exemplu, în niciunul din cazurile de test de mai jos nu se generează eroare:

```
1 razvan@anaconda:~/uso/scripting$ b=3
2
3 razvan@anaconda:~/uso/scripting$ test "a" = $b
4
5 razvan@anaconda:~/uso/scripting$ test 1 -eq $b
```

Numele variabilei poate fi încadrat între acolade. Acestea sunt opțiunale dar sunt utile pentru a proteja numele variabilei de caracterele următoare. În exemplul de mai jos utilizarea acoladelor duce la afișarea dorită, altfel nu ar fi fost afișat nimic, variabila `ba` nefiind inițializată:

```
razvan@anaconda:~/uso/scripting$ echo ${b}a
3a
```

Folosind caracterul `$` se pot face diverite expandări asupra variabilei:

- determinarea numărului de litere compun variabila:

```
razvan@anaconda:~/uso/scripting$ var="alfanumeric"

razvan@anaconda:~/uso/scripting$ echo ${#var}
11
```

- porțiuni din numele variabilei:

```
razvan@anaconda:~/uso/scripting$ echo ${var:3}
anumeric

razvan@anaconda:~/uso/scripting$ echo ${var:3:6}
anumer
```

Expandarea unei comenzi

Expandarea unei comenzi înseamnă executarea acesteia și reținerea rezultatului comenzii. Acest lucru se poate face în două moduri:

```
$(comanda)
```

sau

```
`comanda`
```

(caracterul folosit este ` – *backquote* – se află pe tasta cu ~ de lânga tasta `1`) Rezultatul unei comenzi poate fi reținut într-o variabilă și folosit ulterior.

Expandarea aritmetică

Expandarea aritmetică presupune evaluarea unei expresii și reținerea rezultatului acesteia. Pentru aceasta se folosește operatorul `$((...))` sub forma `$(( expresie ))`.

Dacă nu este nevoie de reținerea rezultatului se poate folosi operatorul `((...))`

```
razvan@anaconda:~/uso/scripting$ a=1

razvan@anaconda:~/uso/scripting$ ((a++))

razvan@anaconda:~/uso/scripting$ echo $a
2
```

12.7.2 Expresii regulate în shell

Shell-ul permite interpretarea expresiilor regulate pentru a permite expandarea numelor de fișiere.

Construcțiile care permit expandarea numelor de fișiere sunt:

- acoladele `{}` permit precizarea unor opțiuni pentru expandare; astfel, expresia `{a,b,c}.d` se va expanda la `a.d b.d c.d`;
- `*` înseamnă expandarea la orice șir de caractere; astfel *.c înseamnă toate fișiere care au extensia `.c`, iar `a*.c` reprezintă toate fișierele care încep cu litera `a` și se termină în `.c`;
- `?` înseamnă potrivirea cu un singur caracter (oricare ar fi acela);
- `[ ... ]` permit specificarea unei clase de caractere; astfel, `[a-z]` înseamnă litere mici `[a-d0-3A-D]` înseamnă literele de la `a` la `d`, de la `A` la `D` sau cifrele de la `0` la `3`; `[^a-z]` înseamnă orice mai puțin literele de la `a` la `z`.

Exemple de utilizare a expresiilor regulate shell sunt prezentate în continuare:

- afișarea tuturor fișierelor C din directorul curent:

```
1 ubuntu@ubuntu:~$ ls *.c
```

- afișarea tuturor fișierelor de tip png, jpg sau gif:

```
1 ubuntu@ubuntu:~$ ls *.{png,jpg,gif}
```

- afișarea tuturor fișierelor al căror nume începe cu 3 cifre:

```
1 ubuntu@ubuntu:~$ ls [0-9][0-9][0-9]*
```

12.8 Parametrii unui script shell

La fel ca orice alt program, un script shell poate primi argumente în linia de comandă. Acestea pot fi folosite pentru a oferi un mod de configurare a scriptului în momentul execuției.

În C, modul în care se realizează accesul la argumentele în linia de comandă este prin intermediul variabilelor `argc` și `argv`. Astfel, variabila `argc` indică numărul de argumente primite în linia de comandă (incluzând numele executabilului), iar `argv` este vectorul de argumente: `argv[0]` este numele executabilului, `argv[1]` numele primului argument, `argv[argc-1]` este ultimul argument.

În Bash, accesul la argumente se realizează prin intermediul variabilei `$#`, reprezentând numărul de argumente, și al variabilelor `$1`, `$2` etc. reprezentând argumentele efective. Ca exemplu, vom folosi scriptul de mai jos, care nu face altceva decât să afișeze argumentele primite:

```
1 razvan@anaconda:~/uso/scripting$ cat -n args.bash
2      1  #!/bin/bash
3      2
4      3  echo "Scriptul are $# argumente."
```

```
5        4  echo "Numele scriptului este $0."
6        5  echo "Argumentele scriptului sunt '$@'."
7        6
8        7  i=1
9        8
10       9  while test $i -le $#; do
11      10          echo "Argumentul $i este ${!i}."
12      11          ((i++))
13      12  done
14
15  razvan@anaconda:~/uso/scripting$ bash args.bash alfa beta gamma delta
16  Scriptul are 4 argumente.
17
18  Numele scriptului este args.bash.
19
20  Argumentele scriptului sunt 'alfa beta gamma delta'.
21
22  Argumentul 1 este alfa.
23
24  Argumentul 2 este beta.
25
26  Argumentul 3 este gamma.
27
28  Argumentul 4 este delta.
```

În cazul exemplului de mai sus, se trimit 4 argumente. Dacă variabila `argc` numără inclusiv executabilul, variabila `$#` contorizează doar argumentele. Variabila `$0` conține numele executabilului. Variabila `$@` conține toate argumentele separate prin spațiu (*blank*) (linia 5). Folosind variabila `i` se parcurg toate argumentele primite în linia de comandă (linia 9) și se afișează (linia 10). Se observă că am folosit construcția `$!i`, pentru a expanda variabila `i` și apoi variabila care reprezintă argumentul primit din linia de comandă.

Un al mod de a realiza parcurgerea argumentelor primite în linia de comandă ar fi fost următorul:

```
1  i=1
2  for arg in $@; do
3          echo "Argumentul $i este $arg."
4          ((i++))
5  done
```

Variabilele `$1`, `$2`, ..., `$N`, reprezentând argumentele primite din linia de comandă, poartă numele de parametri poziționali.

12.8.1 Comanda shift

Comanda **`shift`**, după cum îi spune numele, este folosită la deplasarea argumentelor din linia de comandă în parametri poziționali. În exemplul de mai jos, se face parcurgerea parametrilor poziționali folosind comanda **`shift`**:

```
1  i=1
2  while ! test -z $1 ; do
3          echo "Argumentul $i este $1."
4          ((i++))
5          shift
```

```
6 done
```

Comanda **shift** este responsabilă cu deplasarea spre stânga a argumentelor unui script shell. Astfel, după rularea comenzii **shift**, se pierde informația despre primul argument, variabila `$#` scade cu 1, variabila `$1` reprezintă al doilea argument, variabila `$2` reprezintă al treilea argument etc.

Comanda **shift** poate primi un argument reprezentând contorul de deplasare. Implicit (fără argument) acesta este 1.

12.8.2 Parametri speciali

După cum a fost prezentat anterior, există variabile speciale folosite de shell pentru prelucrarea argumentelor în linia de comandă:

- `$#` reprezintă numărul de argumente din linia de comandă;
- `$0` reprezintă numele scriptului shell;
- `$1`, `$2`, ..., `$N` reprezintă argumentele din linia de comandă;
- `$@` reprezintă lista de argumente separate prin spații.

În afara acestor variabile mai există câțiva parametri speciali, prezentați în continuare:

- `$?` reprezintă valoarea de retur a ultimei comenzi;
- `$$` reprezintă pid-ul procesului bash curent;
- `$!` reprezintă pid-ul ultimului proces lansat în fundal.

12.8.3 Exemplu de utilizare a parametrilor

Un exemplu util este un script shell care să ușureze lucrul cu comanda **find** în momentul căutării unor fișiere după nume. Scriptul este prezentat în continuare:

```
1  #!/bin/bash
2
3  if test $# -ne 2; then
4          echo "Two␣arguments␣are␣required."
5          exit 1
6  fi
7
8  if ! test -d $1; then
9          echo "First␣argument␣is␣not␣a␣directory."
10         exit 1
11 fi
12
13 find "$1" -type f -name "$2"
```

Listing 12.4: Fișierul `myfind.bash`

```
1 razvan@anaconda:~/uso/scripting$ bash myfind.bash . "*.bash"
2 ./while10.bash
3 ./hw.bash
4 ./for_ls.bash
```

```
5 ./service_ip.bash
6 ./while_cat.bash
7 ./rss.bash
8 [...]
```

Se observă că se transmit ca argumente directorul în care se face căutarea și expresia regulată asociată numelui fișierului. În afara apelării **`find`** (linia 13), se verifică numărul de argumente (liniile 3-6) și dacă primul argument este un director (liniile 8-11).

12.9 Funcții

La fel ca majoritatea limbajelor de programare, shell scriptingul permite folosirea de funcții. În general funcțiile sunt bucăți de cod folosite pentru a separa o funcționalitate sau pentru refolosire (pentru a nu se scrie de două ori același cod). De asemenea, funcțiile au un rol important în mărirea lizibilității codului.

În shell scripting, funcțiile intervin în cazul scripturilor de dimensiune mai mare, unde există bucăți de cod care sunt folosite de mai multe ori.

12.9.1 Sintaxa unei funcții

Prezentăm sintaxa comenzii prin următorul exemplu, în care se poate șterge sau adăuga un utilizator în sistem:

```
1  #!/bin/bash
2
3  if test $# -ne 1; then
4          echo "One␣argument␣required:␣a␣or␣d␣(add␣or␣delete)."
5          exit 1
6  fi
7
8  username=testuser
9  groupname=testgroup
10
11 function add ()
12 {
13         cat /etc/group | grep $groupname &> /dev/null
14         if test $? -eq 1; then           # no group
15                 addgroup $groupname
16                 echo "Group␣$groupname␣didn't␣exist␣so␣it␣was␣created."
17         fi
18         useradd -m -d /home/$username -g $groupname -s /bin/bash
              $username
19         echo "User␣$username␣added.␣Use␣passwd␣to␣change␣password."
20 }
21
22 function delete ()
23 {
24         userdel -r $username
25         echo "User␣$username␣deleted."
26 }
27
28 case $1 in
```

```
29          "a") add;;
30          "d") delete;;
31          *) echo "Required arguments are 'a' or 'd'."; exit 1;;
32  esac
```

Listing 12.5: Fișierul `user.bash`

```
1  root@anaconda:/home/razvan/uso/scripting# bash user.bash a
2  Adding group 'testgroup' (GID 1031) ...
3
4  Done.
5
6  Group testgroup didn't exist so it was created.
7
8  User testuser added. Use passwd to change password.
9
10 root@anaconda:/home/razvan/uso/scripting# bash user.bash d
11 User testuser deleted.
```

În exemplul de mai sus funcțiile sunt `add` și `delete` și sunt folosite, respectiv, pentru a adăuga și a elimina un utilizator în/din sistem. Sintaxa unei definiții de funcție este:

```
1  function nume_functie ()
2  {
3    cod shell
4  }
```

Se observă că o funcție shell are un antet și un corp. În antet se prezintă numele funcției (`nume_functie`), iar în corp se regăsesc comenzile ce se execută în momentul apelării funcției. Cuvântul `function` este cuvânt rezervat și poate fi omis. Astfel, o funcție definită:

```
1  function testare ()
2  {
3    ...
4  }
```

poate fi definită și astfel:

```
1  testare ()
2  {
3    ...
4  }
```

Pentru a apela o funcție se folosește direct numele funcției urmat de parametri, exact ca în cazul apelului unei comenzi.

12.9.2 Parametrii unei funcții

În mod evident, o funcție poate primi și argumente. Modul de definire funcției nu se schimbă; altfel spus, într-un shell script o funcție nu are parametri formali. Totuși, modul de apel se schimbă prin precizarea parametrilor pentru funcție.

Pentru a exemplifica modul în care se transmit parametrii unei funcții vom folosi un exemplu de calcul a sumei unor numere naturale transmise ca argumente funcției `sum-func`:

```
#!/bin/bash

function sum_func ()
{
        sum=0
        while ! test -z $1; do
                sum=$(($sum + $1))
                shift
        done
}

sum_func 1 2 3 4 5
echo "Sum: $sum"

sum_func 2 3 5 7 11 13 17 19
echo "Sum: $sum"
```

Listing 12.6: Fișierul `sum-func.bash`

```
razvan@anaconda:~/uso/scripting$ bash sum-func.bash
Sum: 15
Sum: 77
```

Se observă că argumentele unei funcții se obțin tot folosind parametri poziționali (`$1`, `$2` etc.). Variabila `$#` reprezintă numărul de argumente transmise funcției. Variabila `$0` rămâne neschimbată și reprezintă numele scriptului. De asemenea, se poate folosi comanda **`shift`**. În momentul în care funcția se întoarce, valorile parametrilor poziționali sunt restaurate. Numele funcției este reținut în variabila `FUNCNAME`.

Variabila `sum` a fost definită în cadrul funcției și este vizibilă și în exterior. Dacă se dorește ca o variabilă să fie vizibilă numai în interiorul unei funcții aceasta trebuie prefixată folosind cuvântul cheie `local`.

12.10 Scripturile de pornire Bash

Scripturile de pornire Bash sunt fișiere intepretate la pornirea unei sesiuni shell. Acestea se ocupă de actualizarea variabilelor de mediu și de rularea unor comenzi personalizate de utilizator; un exemplu îl constituie configurarea de alias-uri.

Sesiunile de shell pot fi clasificate în:

- sesiuni **interactive/non-interactive** după cum permit sau nu rularea de comenzi prin prezentarea unui prompt utilizatorului; o sesiune non-interactivă este, de cele mai multe ori, folosită pentru rularea unui script shell
- sesiuni de **autentificare** (*login*)/**non-autentificare** (*non-login*); sesiunile de emulator de terminal (**`gnome-terminal`**, **`konsole`**, **`xterm`**) sunt sesiuni *non-login*

Fișierele de pornire sunt interpretate de sesiunile de shell interactive în modul următor:

- o sesiune de shell *login* va interpreta, în ordine, fișierele `/etc/profile`, apoi `~/.bash_profile`, `~/.bash_login` și `~/.profile`; la încheierea sesiunii se interpretează fișierul `~/.bash_logout`

- o sesiune de shell *non-login* va interpreta comenzile din `~/.bashrc`

  Fișierul `~/.bashrc` este fișierul recomandat pentru personalizarea sesiunilor de shell ale unui utilizator. Întrucât un shell de login nu interpretează `~/.bashrc`, fișierul `~/.bash_profile` va conține, de obicei, o linie de forma:

```
1 # include .bashrc if it exists
2 if [ -f ~/.bashrc ]; then
3     . ~/.bashrc
4 fi
```

12.10.1 Variabile de mediu

După cum a fost specificat anterior, unul din rolurile importante ale scripturilor de pornire shell este configurarea variabilelor de mediu.

Variabilele de mediu sunt variabile care afectează modul în care un proces rulează. În mod obișnuit, în sistemele Unix, un proces moștenește variabilele de mediu ale părintelui său. Astfel, pentru procesul asociat execuției unei comenzi, variabilele de mediu sunt cele ale shell-ului.

De cele mai multe ori, variabilele de mediu sunt configurate în scripturile de pornire ale Bash. Declarația unei variabile de mediu se face în formatul

```
1 export VARIABLE=value
```

Dacă nu se precizează **`export`**, atunci variabila nu va fi moștenită de procesele fiu. În mod obișnuit, variabilele de mediu sunt denumite cu majuscule.

Vizualizarea variabilelor de mediu existente la un moment dat se realizează cu ajutorul comenzii **`env`** sau **`set`**:

```
1 razvan@anaconda:~/uso/scripting$ env
2 SSH_AGENT_PID=7985
3 TERM=xterm
4 SHELL=/bin/bash
5  [...]
```

Exemple de variabile de mediu sunt:

- `PATH` reține o listă de directoare folosite de shell pentru a găsi executabilele asociate comenzii; lista este separată prin `:` (două puncte); dacă se dorește adăugarea directorului `~/bin` la variabila de mediu, se folosește comanda:

```
1 razvan@anaconda:~/uso/scripting$ echo $PATH
2 /usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
3
4 razvan@anaconda:~/uso/scripting$ export PATH="$PATH":~/bin
5
6 razvan@anaconda:~/uso/scripting$ echo $PATH
7 /usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games:/home/razvan/
  bin
```

- `HOME` reține directorul de bază al utilizatorului curent
- `USER` este numele utilizatorului curent
- `EDITOR` este editorul implicit

12.11 Studii de caz

12.11.1 Contorizarea numărului de utilizatori autentificați în sistem

Folosind cunoștințele din acest capitol, se poate rezolva corect problema contorizării numărului de utilizatori autentificați în ziua de *luni* în sistem, problemă propusă în secțiunea 12.3.4. Soluția este prezentată în continuare:

```
1  #!/bin/bash
2
3  user_day="Monday"
4
5  case $user_day in
6          "Monday") day_last="Mon";;
7          "Tuesday") day_last="Tue";;
8          "Wednesday") day_last="Wed";;
9          "Thursday") day_last="Thu";;
10         "Friday") day_last="Fri";;
11         "Saturday") day_last="Sat";;
12         "Sunday") day_last="Sun";;
13         *) echo "Zi invalida"; exit 1;;
14 esac
15
16 last | head -n -2 | while read user term ip day other; do
17         if test $day = $day_last; then
18                 echo "$user";
19         fi
20 done | sort -u | wc -l
```

Listing 12.7: Fișierul `count-login.bash`

```
1  razvan@anaconda:~/uso/scripting$ bash count-login.bash
2  9
```

Linia 3 definește variabila `user_day` folosită pentru a reține ziua despre care dorim informații (generalizat). Aceasta va fi dată în formatul complet pentru a avantaja utilizatorul. Liniile 5-14 se ocupă de traducerea zilei în formatul **`last`**; ziua în formatul **`last`** (prescurtat) este reținută în variabila `day_last`.

Liniile 16-20 reprezintă, de fapt, o comandă înlănțuită. Componentele lanțului de comenzii sunt:

- **`last`** – afișează informații despre utilizatorii autentificați în sistem;
- **`head -n -2`** – reține primele linii de la intrare, mai puțin ultimele două; este nevoie de acest lucru deoarece ultimele două linii date la ieșire de **`last`** nu sunt utile;
- ciclul `while` este folosit pentru a citi, folosind **`read`**, diversele câmpuri ale fiecărei linii date de last: numele de utilizator, terminalul asociat, adresa IP de conectare, ziua în care s-a realizat autentificarea; variabila `other`, fiind ultima variabilă, reține restul de informații până la sfârșitul liniei (informații care nu interesează); dacă ziua corespunde zilei reținute în variabila `day_last` atunci se afișează la ieșire numele de utilizator;

- **`sort -u`** – sortează și reține în mod unic liniile primite la intrare; în cazul nostru aceste linii sunt chiar cele afișate de ciclul `while`, adică utilizatorii care s-au autentificat în sistem în ziua dată de variabila `user_day`;
- **`wc -l`** – contorizează numărului de linii furnizat de **`sort -u`**; rezultatul este numărul de utilizatori distincți care s-au autentificat în sistem în ziua de dată de variabila `user_day`.

12.11.2 Schimbarea promptului shell

Promptul este șirul de caractere oferit de shell unde utilizatorul poate introduce comenzi. În Bash, un prompt care se termină în caracterul `$` reprezintă o sesiune de shell a unui utilizator neprivilegiat; un prompt care se termină în caracterul `#` este asociat unei sesiuni de shell a utilizatorului `root`.

Aspectul promptului este controlat prin intermediul unei variabile speciale, anume `PS1`. Aceasta controlează modul în care promptul este oferit utilizatorului. Comanda **`echo`** permite inspectarea variabilei:

```
1 razvan@anaconda:~$ echo $PS1
2 \u@\h:\w\$
```

Se observă că variabila cuprinde o serie de caractere speciale citate cu ajutorul *backslash*:

- `\u` înseamnă numele utilizatorului; în exemplul de mai sus se traduce în șirul `razvan`
- `\h` înseamnă numele sistemului
- `\w` înseamnă afișarea directorului curent, cu abrevierea directotului de bază la caracterul `~`
- `\$` înseamnă `#` pentru `root` sau `$` în rest

Promptul poate fi schimbat la un șir de caractere simplu:

```
1 razvan@anaconda:~$ PS1="prompt: "
2 prompt:
```

Promptul fi personalizat și folosind o serie de caractere speciale ca în exemplele de mai jos:

```
1  prompt: echo PS1="\u-\h[\d]\$ "
2  PS1=\u-\h[\d]$
3
4  prompt: PS1="\u-\h[\d]\$ "
5
6  razvan-anaconda[Fri Sep 14]$ PS1="\u+\h \W (\A)"
7
8  razvan+anaconda ~ (18:08)cd uso/
9
10 razvan+anaconda uso (18:08)cd ~/uso/conv/
11
12 razvan+anaconda conv (18:09)
```

Secvențe speciale care au fost folosite în exemplele de mai sus sunt:

- `\d` înseamnă data curentă
- `\A` înseamnă timpul curent
- `\W` înseamnă directorul curent ca și cale relativă față de directorul home (\w afișează calea completă)

O listă completă cu secvențele speciale de personalizare a promptului se poate găsi online[1].

Variabila `PS1` este definită în `/etc/profile` și `~/.bashrc`. Pentru configurarea permanentă a acesteia se recomandă utilizarea fișierului `~/.bashrc`.

Folosirea de culori în prompt

Promptul poate fi colorat prin folosirea secvenței speciale `\e[`. Astfel, dacă se dorește afișarea unui prompt de culoare roșie, se folosește comanda:

```
1 razvan@anaconda:~/uso/conv$ PS1="\e[0;31m\u@\h:\w\$ \e[m"
```

Pentru obținerea unui prompt de culoare verde comanda este:

```
1 razvan@anaconda:~/uso/conv$ PS1="\e[0;32m\u@\h:\w\$ \e[m"
```

Secvențele speciale `\e[x;ym` și `\e[m` definesc spațiul de aplicabilitate al unei culori. Secvența `x;y` definește culoarea utilizată. Secvențe posibile sunt:

- `0;30` – negru
- `0;31` – roșu
- `0;32` – verde
- `0;33` – galben
- `0;34` – albastru
- `0;35` – magenta
- `0;36` – cyan
- `0;37` – alb

În lista de mai sus `x` are valoare `0`. Se poate folosi valoarea `1` pentru versiunea deschisă a culorii. În mod evident, se pot asocia culori diferite pentru componentele promptului, ca în exemplul de mai jos:

```
1 razvan@anaconda:~$ PS1="\e[0;36m\u\e[m@\e[0;34m\h\e[m:\e[0;31m\w\e[m\$ "
```

În exemplul de mai sus, în cadrul promptului, numele utilizatorului va fi afișat folosind cyan, numele sistemului folosind albastru, iar numele directorului curent folosind roșu.

[1] http://tldp.org/HOWTO/Bash-Prompt-HOWTO/bash-prompt-escape-sequences.html

12.11.3 Batch scripting în Windows

În Windows, scripturile shell au ca echivalent *scripturi batch*. Un script batch are de obicei extensia `.bat`. Programul folosit pentru rularea scripturilor batch este **`cmd`** (interpretorul de comenzi pe Windows).

La fel ca un script shell, un script batch permite execuția de comenzi native Windows și programarea cu ajutorul variabilelor, a instrucțiunilor de decizie și a instrucțiunilor de ciclare.

Comenzi Windows

Rularea unei comenzi în Windows se realizează prin intermediul interpretorului de comenzi Windows. Pentru pornirea acestuia, în Windows XP trebuie pornită aplicația **Command Prompt**: *Start->All Programs->Accesories->Command Prompt*.

În afara comenzilor obișnuite pentru lucrul cu sistemul de fișiere (**`dir`**, **`cd`**, **`mkdir`**, **`rmdir`**, **`del`**, **`copy`**, **`move`** etc.), există și alte comenzi utile. Lista cu toate comenzile Windows se poate obține folosind comanda **`help`**:

```
C:\Documents and Settings\Razvan.RAGNAROK>help
For more information on a specific command, type HELP command-name
ASSOC    Displays or modifies file extension associations.

AT       Schedules commands and programs to run on a computer.

ATTRIB   Displays or changes file attributes.

BREAK    Sets or clears extended CTRL+C checking.

CACLS    Displays or modifies access control lists (ACLs) of files.

[...]
```

În tabelul 12.2 este prezentat un tabel cu comenzile utile Linux și echivalentul lor în Windows:

Operatori

Interpretorul de comenzi Windows are, asemănător interpretorului Bash, operatori de redirectare sau de comunicare între procese. În exemplul de mai jos dorim să reținem în fișierul `del.txt` informații despre comenzile care se ocupă cu ștergerea de informații:

```
C:\Documents and Settings\Razvan.RAGNAROK>help | findstr "[Dd]elete" >
del.txt

C:\Documents and Settings\Razvan.RAGNAROK>type del.txt
DEL      Deletes one or more files.

ERASE    Deletes one or more files.

LABEL    Creates, changes, or deletes the volume label of a disk.

```

```
10 [...]
```

Variabile

Ca și în Bash, variabilele nu au un tip. Definirea acestora se realizează cu ajutorul comenzii **`set`**:

```
1 C:\Documents and Settings\Razvan.RAGNAROK>set a=3
2
3 C:\Documents and Settings\Razvan.RAGNAROK>echo %a%
4 3
```

Afișarea conținutul unei variabile nu se mai realizează prefixând-o cu ajutorul caracterului `$`, ci folosind caracterul `%` înainte și după numele variabilei.

Ca și în Bash, comanda **`set`** este folosită și pentru afișarea variabilelor curente:

```
1  C:\Documents and Settings\Razvan.RAGNAROK>set
2  a=3
3  ALLUSERSPROFILE=C:\Documents and Settings\All Users
4  APPDATA=C:\Documents and Settings\Razvan.RAGNAROK\Application Data
5  CommonProgramFiles=C:\Program Files\Common Files
6  COMPUTERNAME=RAGNAROK
7  ComSpec=C:\WINDOWS\system32\cmd.exe
8  FP_NO_HOST_CHECK=NO
9  HOMEDRIVE=C:
10 HOMEPATH=\Documents and Settings\Razvan.RAGNAROK
11 [...]
```

Variabile de mediu Dacă în Bash variabilele de mediu sunt definite în fișierele de pornire, în Windows trebuie folosită o fereastră de configurare. Pentru accesul la aceasta se folosește: click dreapta pe *My Computer->Properties->Advanced->Environment Variables.*

Variabila **`PATH`** deține, ca și în cazul Bash, lista de directoare ale căror executabile pot fi rulate din linia de comandă. Spre deosebire de Bash, separatorul folosit este `;` (punct și virgulă).

În exemplul de mai jos se adaugă directorul `D:\mybin` la calea de executabile:

```
1 C:\Documents and Settings\Razvan.RAGNAROK>echo %PATH%
2 c:\program files\imagemagick-6.3.4-q16;C:\Program Files\MiKTeX 2.6\miktex
  \bin;C:
3 \WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;D:\Software\
  swftools
4
5 C:\Documents and Settings\Razvan.RAGNAROK>set PATH=%PATH%;D:\mybin\
6
7 C:\Documents and Settings\Razvan.RAGNAROK>echo %PATH%
8 c:\program files\imagemagick-6.3.4-q16;C:\Program Files\MiKTeX 2.6\miktex
  \bin;C:
9 \WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;D:\Software\
  swftools;D:\mybin\
```

Programarea batch

După cum s-a precizat, echivalentul scripturilor shell din Linux sunt scripturile batch din Windows. De obicei, acestea au extensia `.bat`. Exemplul cel mai simplu de script batch este prezentat mai jos:

```
C:\DOCUME~1\RAZVAN~1.RAG>type hw.bat
@echo off
echo "Hello, World!"

C:\DOCUME~1\RAZVAN~1.RAG>hw.bat
"Hello, World!"
```

În mod implicit, rularea unui script batch duce la afișarea comenzilor întâlnite (echivalent cu `set -x` din Bash). Pentru a preîntâmpina acest lucru se folosește directiva `@echo off`.

Un emulator al comenzii **`dir`** folosește `for` și este prezentat în continuare:

```
C:\DOCUME~1\RAZVAN~1.RAG>type dir.bat
@echo off
for %%f in (*) do echo %%f

C:\DOCUME~1\RAZVAN~1.RAG>dir.bat
.gtk-bookmarks
del.txt
dir.bat
gsview32.ini
hw.bat
logfile.txt
PUTTY.RND
winscp.RND
_viminfo
```

Instrucțiunea `if` are o sintaxa asemănătoare instrucțiunii echivalente din Bash:

```
C:\DOCUME~1\RAZVAN~1.RAG>set a=3

C:\DOCUME~1\RAZVAN~1.RAG>if %a% EQU 3 echo "true"
"true"
```

Mai multe despre comenzile instrucțiunile if și for se pot afla prin consultarea informației de ajutor: `if /?` și `for /?`.

12.11.4 PowerShell pe Windows

PowerShell este un nou interpretor de comenzi dezvoltat de Microsoft. Se bazează pe programare orientată obiect și pe versiunea 2.0 a platformei de dezvoltare .NET. Windows PowerShell este gratis; poate fi descărcat de pe site-ul asociat[1].

Windows PowerShell urmează, într-o oarecare măsură, ideea din spatele utilitarelor shell Unix, anume de a realiza sarcini complexe prin combinarea unor componente deja existente. Aceste componente sunt denumite cmdlets și sunt, de fapt, instanțe ale unor clase .NET. Deosebirea de shellurile Unix este faptul că în loc de a crea o bandă de transmitere a informației în format text (pipeline), cmdlet-ii comunică folosind obiecte.

[1] http://www.microsoft.com/windowsserver2003/technologies/management/powershell/default.mspx

Cmdlets

Cmdlet-ii reprezintă blocurile de bază pentru PowerShell echivalent cu comenzile de bază Unix folosite de Bash.

Astfel, pentru listarea proceselor din sistem se folosește cmdlet-ul **`get-process`**. Acesta prezintă aliasul **`ps`** și este echivalent comenzii **`ps`** din Linux. Pentru listarea conținutului unui director se folosește cmdlet-ul **`get-childitem`** cu aliasul **`ls`**.

Dacă, spre exemplu, un utilizator dorește omorârea proceselor al căror nume începe cu litera `p`, se folosește comanda:

```
PS> get-process p* | stop-process
```

Folosind concepte din programarea orientată obiect, se pot extrage câmpuri din rezultatul unei comenzi. Astfel, dacă se dorește aflarea timpului ultimului acces pentru un directorul `Cookies`, se folosește comanda:

```
PS> $(get-item Cookies).lastaccesstime
```

Pentru afișarea proceselor din sistem în formă tabelară se poate folosi comanda

```
PS> Get-Process | Format-Table Id, Name
  Id Name
  -- ----
2876 alg
 532 ApntEx
2044 Apoint
3448 calc
1824 CFSvcs
```

Variabile și structuri de control

PowerShell permite crearea de scripturi. Ca și Bash, printre facilitățile precizate se numără completare automată, utilizarea de variabile, structuri de control și funcții.

Spre deosebire de Bash, variabilele PowerShell au un tip:

```
PS C:\> $b = 3.1415926

PS C:\> $b
3.1415926

PS C:\> $b.GetType().Name
```

În Windows PowerShell, echivalentul pentru `for` (din Bash) este `foreach`:

```
PS C:\WINDOWS\system32> 1..10 | foreach { $_ * 2 }
2
4
6
8
10
12
14
16
18
20
```

Informații complete despre Windows PowerShell se găsesc în documentația care sosește cu pachetul de instalare.

Informații complete și tutoriale sed se pot găsi la online[1].

Cuvinte cheie

- shell
- bash
- script shell
- readline
- editarea comenzilor
- istoricul comenzilor
- reverse search
- autocompletion
- comenzi interne (built-in)
- comenzi externe
- shebang
- echo
- printf
- set
- exit
- read
- redirectarea intrării/ieșirii/erorii standard
- operatorii <, >, 2>, &>, >>
- here document, here string
- operatorii <<, <<<
- operatorii de secvențiere ;, &&, ||
- operatorul de comunicație |
- variabile shell
- caractere speciale shell: $, #, *, ", ‘ , \, (,), , , &, |, ;, [,], ?, blank
- if, test, comanda de test [...]
- for, while, until
- filtre de text: cat, tac, nl, sort, uniq, head, tail, cut, tr, grep
- expresie regulată (pattern matching)
- xargs, locate, find
- expandarea shell: expandarea variabilei, substituția comenzii, expandare aritmetică
- parametrii unui script shell; expresiile $0, $n, $#, $@, $?, $$, $!
- funcții shell
- scripturi de inițializare: .bashrc, .bash_profile, /etc/bash_profile
- variabile de mediu
- batch scripting
- power shell; cmdlet
- sed

Întrebări

1. Care din următoarele nu este un intepretor de comenzi?
   - ❑ Bash
   - ❑ PowerShell
   - ❑ Command Prompt

[1] http://sed.sourceforge.net/grabbag/tutorials/

- ❑ Flash

2. Ce comandă este folosită pentru căutarea unei expresii regulate într-un fișier?
   - ❑ tr
   - ❑ grep
   - ❑ cut
   - ❑ sort
3. Care din următorii operatori este asociat cu un *here document*?
   - ❑ `>`
   - ❑ `|`
   - ❑ `&>`
   - ❑ `<<`
4. Care din următoarele opțiuni testează egalitatea între două șiruri?
   - ❑ `set $a == $b`
   - ❑ `[$a == $b]`
   - ❑ `test $a == $b`
   - ❑ `{ $a == $b }`
5. Care este efectul comenzii **`sort -n file.txt`**?
   - ❑ sortează liniile nevide din fișier
   - ❑ sortează liniile vide din fișier
   - ❑ sortează liniile unice din fișier
   - ❑ sortează liniile în format numeric
6. Ce utilitar NU va ajuta un utilizator să substituie caracterul `a` cu `A` într-un fișier?
   - ❑ **`vim`**
   - ❑ **`cut`**
   - ❑ **`sed`**
   - ❑ **`tr`**
7. Care din următoarele NU este un avantaj al unui shell script?
   - ❑ viteza mare de execuție
   - ❑ automatizarea sarcinilor
   - ❑ folosirea de componente existente
   - ❑ depanare facilă
8. Care este utilitatea *shebang* (`#!`)?
   - ❑ permite editarea unui script shell

- ❑ permite asocierea de drepturi de execuție unui script shell
- ❑ prezintă calea către utilitarul folosit pentru rularea scriptului
- ❑ este o linie de comentariu specială într-un script shell

9. Care din următoarele NU este o instrucțiune în Bash?
   - ❑ `for`
   - ❑ `while`
   - ❑ `until`
   - ❑ `repeat`
10. Care din următoarele comenzi NU afișează intrările din directorul curent?
   - ❑ `ls`
   - ❑ `for i in *; do echo $i; done`
   - ❑ `find .  -mindepth 2 -type f`
   - ❑ `ls *`

Tabelul 12.1: Condiții în shell

operator	tip operanzi	condiție verificată	utilizare
`-e`	1 fișier	fișierul există	`-e fisier`
`-f`	1 fișier	fișierul este fișier obișnuit (regular file)	`-f fisier`
`-s`	1 fișier	fișierul are dimensiune non-zero	`-s fisier`
`-d`	1 fișier	fișierul este director	`-d fisier`
`-r`	1 fișier	fișierul poate fi citit	`-r fisier`
`-w`	1 fișier	fișierul poate fi scris	`-w fisier`
`-x`	1 fișier	fișierul poate fi executat	`-x fisier`
`-O`	1 fișier	fișierul este deținul de utilizator	`-O fisier`
`-G`	1 fișier	grupul ce deține fișierul este și grupul utilizatorului	`-G fisier`
`-nt`	2 fișiere	primul fișier este mai recent decât al doilea	`fisier1 -nt fisier2`
`-ot`	2 fișiere	primul fișier este mai puțin recent decât al doilea	`fisier1 -ot fisier2`
`-eq`	2 numere	numerele sunt egale	`numar1 -eq numar2`
`-ne`	2 numere	numerele nu sunt egalee	`numar1 -nq numar2`
`-gt`	2 numere	primul număr este mai mare decât al doilea	`numar1 -gt numar2`
`-ge`	2 numere	primul număr este mai mic sau egal cu al doilea	`numar1 -ge numar2`
`-lt`	2 numere	primul număr este mai mic decât al doilea	`numar1 -lt numar2`
`-le`	2 numere	primul număr este mai mic sau egal cu al doilea	`numar1 -le numar2`
`=`	2 șiruri de caractere	șirurile sunt identice	`sir1 = sir2`
`!=`	2 șiruri de caractere	șirurile nu sunt identice	`sir1 != sir2`
`-z`	1 șir de caractere	șirul este vid	`-z sir`

Tabelul 12.2: Comenzi utile Linux și Windows

Linux	Windows	Efect
pwd	cd	afișează directorul curent
ls	dir	listează conținut director
cd	cd	schimbă director
mkdir	mkdir, md	creează director
rmdir	rmdir, rm	șterge director
touch	–	creează fișier
rm	del, erase	șterge fișier
cp	copy	copiază fișier
mv	move	mută fișier
cat	type	afișează conținut fișier
echo	echo	afișează mesaj
less, more	more	afișare paginată fișier
cmp	comp	compară fișiere
diff	fc	compară și afișează diferențe
file	ftype	afișează tipul fișierului
grep	findstr	caută în fișier
shift	shift	schimbă parametrii poziționali
sort	sort	sortează intrarea

Capitolul 13

Mediul grafic

A bus station is where a bus stops. A train station is where a train stops. On my desk I have a workstation...
(Anonymous)

Ce se învață din acest capitol?

- Tipuri de imagini, fonturi, Unicode, UTF-8
- Interfața grafică în Linux: X Window System
- Componentele X Window System: window manager, desktop environment, display manager
- Configurări de bază: rezoluția, aranjamentul caracterelor pe tastatură
- Configurarea sistemului folosind KDE
- Servicii desktop
- Configurare X peste SSH, VNC, RDC

În momentul de față, cea mai utilizată metodă de interacțiune cu sistemul de operare este interfața GUI. Principalul său avantaj față de celelalte tipuri de interfețe (CLI, TUI) (vezi și secțiunea 2.3.1) constă în ușurința cu care utilizatorii învață să o folosească.

Evoluția interfețelor grafice s-a realizat în paralel cu evoluția hardware-ului. Prima încercare de realizare a unei interfețe grafice a avut loc în anii '60 la *Institutul de Cercetări al Universității Standford*[1]. La acel moment, interfața grafică cuprindea un mouse elementar cu ajutorul căruia se puteau accesa link-uri în documentele text (*text hyperlinks*).

În momentul de față, interfețele grafice încep să păsească în era 3D. Aero din Windows Vista/7[2], Compiz[3]/KWin[4] din Linux și Quartz Compositor[5] din MacOS X oferă

[1] Stanford Research Institute – http://www.sri.com/

[2] http://www.microsoft.com/windows/windows-vista/features/aero.aspx
http://www.microsoft.com/windows/windows-7/features/aero.aspx

[3] http://en.wikipedia.org/wiki/Compiz

[4] http://en.wikipedia.org/wiki/KWin

[5] http://arstechnica.com/apple/reviews/2005/04/macosx-10-4.ars/13

utilizatorului o experiență 3D la a cărei realizare participă în mare măsură placa video și nu procesorul ca până acum.

13.1 Concepte în mediul grafic

13.1.1 Tipuri de imagini

În funcție de modul de stocare a informației, imaginile digitale se impart în două tipuri: *imagini raster* (bitmap) și *imagini vectoriale*.

> O imagine **raster** (*bitmap*) reține informația ca o matrice de puncte.

De exemplu, fotografiile realizate cu o cameră foto digitală sunt imagini raster. Este formatul de imagini cel mai des folosit, având diverse implementări: `BMP`, `JPEG`, `GIF`, `PNG` etc. Dezavantajul acestui tip de imagini este că, atunci când imaginea este redimensionată (scalată), conținutul noii imagini trebuie aproximat din conținutul imaginii inițiale. Această aproximare se poate realiza cu rezultate bune în cazul în care imaginea este micșorată. În cazul în care imagine este mărită, rezultatele sunt cu atât mai nesatisfăcătoare cu cât factorul de scalare este mai mare, precum se poate vedea în figura 13.1.

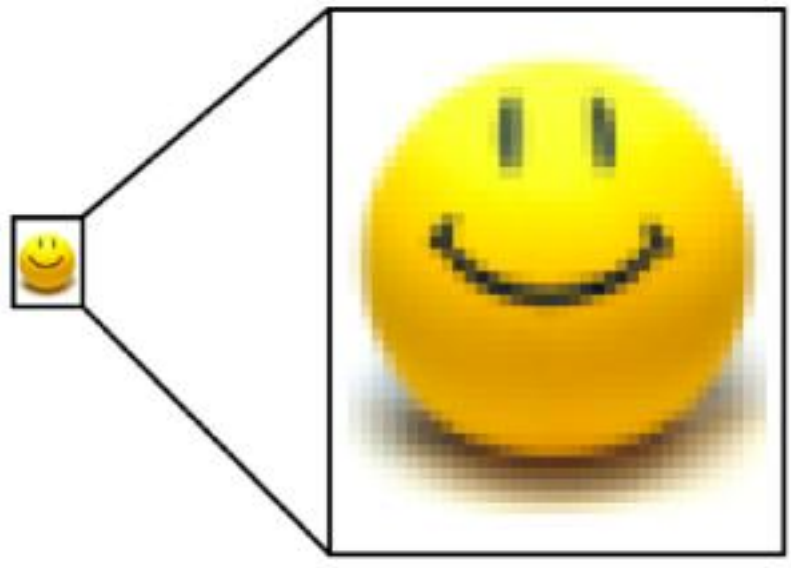


Figura 13.1: Mărirea unei imagini în format raster

Fiecare punct dintr-o imagine raster (bitmap) este descris prin mai multe componente, în funcție de suportul de ieșire:

- **pe ecran** un punct dintr-o imagine raster este descris prin 3 valori reprezentând intensitatea luminoasă (pe 8 biți) a următoarelor culori: *roșu* (Red), *verde* (Green), *albastru* (Blue) – prescurtate RGB. În cazuri speciale (prelucrări profesionale de imagini), intensitatea fiecărei culori poate fi exprimată pe 16 biți (crescând astfel spectrul de culori ce poate fi reprezentat). De asemenea, pe langă cele 3 componente (RGB), anumite imagini digitale (precum formatul PNG) conțin și o a 4-a componentă numită *transparență* (Alpha). Astfel, în cazul imaginilor cu transparență (RGBA), pentru fiecare punct se asociază și un nivel de transparență, permițând astfel ascunderea (totală sau parțială) a unor părți din imagine.

- **pe suport tipărit**, un punct dintr-o imagine raster este descris prin 4 valori, reprezentând cantitatea de culoare necesară pentru obținerea culorii punctului din patru componente: cyan (C), magenta (M), galben (Yellow), negru (black – K)

Într-o imagine **vectorială** informațiile sunt reținute sub formă elemente geometrice de bază: linii, elipse, puncte și gradienți de culoare.

Avantajul major al acestor tipuri de imagini este acela că sunt foarte ușor de redimensionat (practic, culoarea fiecărui punct din imagine poate fi calculată exact pe baza elementelor ce compun imaginea). În urma redimensionării (micșorare sau mărire), calitatea imaginii nu se pierde (figura 13.2). Există câteva formate mai cunoscute care stochează conținutul în format vectorial, cele mai cunoscute fiind: `SVG`[1], `PS`, `EPS`, `PDF`.

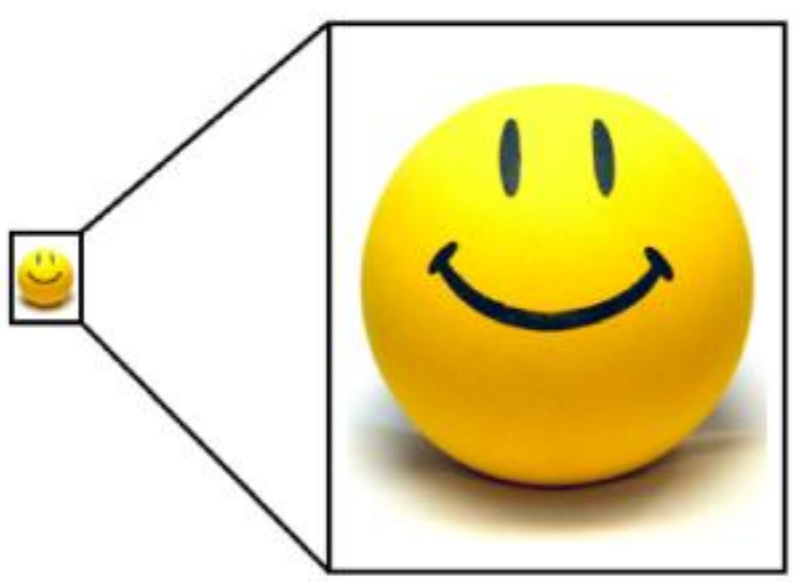


Figura 13.2: Mărirea unei imagini în format vectorial

13.1.2 Fonturi. Unicode

Fonturile sunt fișiere într-un anumit format care conțin informații despre modul de desenare a caracterelor pe ecran. Printre cele mai utilizate în prezent formate pentru fonturi se numără *True Type* (`.ttf`) și *Open Type* (`.otf`).

Fonturile sunt de cele mai multe ori descrieri vectoriale ale aspectului caracterelor, motiv pentru care ele pot scala atât la dimensiuni foarte mici cât și la dimensiuni foarte mari fără pierderi calitative. Exista 4 stiluri mai cunoscute în care se poate prezenta un font: *normal*, *italic* (caractere aldine – caracterele sunt ușor înclinate spre dreapta), *bold* (caracterele sunt îngroșate), *bold-italic* (caractere aldine îngroșate).

Inițial, setul de caractere ASCII[2] a necesitat 7 biți pentru reprezentare (un bit fiind folosit pentru calculul parității în transmiterea fiecărui caracter). Acest lucru permitea existența a 128 caractere, 33 dintre acestea fiind caractere de control. Ulterior, când bitul 8 nu a mai fost folosit ca bit de paritate, folosirea acestuia a dus la extinderea setului de caractere cu încă 128. Pe cele 128 de noi poziții au fost plasate caractere și simboluri dar s-a constatat necesitatea unor extensii.

[1] acceptat de W3C ca standard pentru WWW – http://www.w3.org/Graphics/SVG/

[2] http://en.wikipedia.org/wiki/ASCII

Astfel, a apărut noțiunea de pagină de coduri (code page)[1] prin care erau definite noile 128 de caractere disponibile. Pentru o perioadă de timp s-au folosit frecvent pagini de coduri create fie pentru o anumită limbă fie pentru grupuri de limbi.

Nevoia de a avea mai mult de 256 caractere accesibile la un moment dat a dus la impunerea unui nou standard de reprezentare a setului de caractere – **Unicode** (în prezent ajuns la versiunea 5). Acesta permite reprezentarea prin până la 4 octeți a unui caracter (folosind maxim 32 de biți). În acest fel sunt acoperite inclusiv caracterele asiatice.

Există mai multe standarde[2] pentru codificarea caracterelor Unicode în fișiere. Cel mai folosit este `UTF-8`, format în care primele 128 de caractere sunt identice cu cele din ASCII. Acesta este un format cu număr variabil de octeți, numărul de octeți ocupați de un caracter fiind dat de numărul de biți de `1`, până la primul bit `0`, din primul octet (începând de la cel mai semnificativ bit[3]). Dacă primul bit este `0`, atunci caracterul ocupă un octet și, în mod special, este acelasi caracter ca în setul ASCII. Dacă un octet are primii biți `10`, atunci este un octet care continuă reprezentarea unui caracter. Mai multe informații despre codificarea `UTF-8` se pot găsi online[4].

În prezent, fonturile importante din sistemele de operare moderne au suport pentru un număr semnificativ de caractere definite în standardul Unicode.

13.2 Interfața grafică în Linux. Componente

13.2.1 X Window System

> **X Window System** (cunoscut ca **X11** sau ca **X**) este un cadru (framework) ce permite dezvoltarea de interfețe grafice în medii Unix.

Monitoarele (CRT sau LCD) afișează imagini în format raster. Din acest motiv, imaginea finală trimisă către acestea este în format raster.

X Window System permite: desenarea și mutarea ferestrelor pe ecran, interacțiunea cu mouse-ul și interacțiunea cu tastatura. Restul este lăsat în seama programelor, motiv pentru care aspectul vizual al interfețelor grafice bazate pe X diferă destul de mult de la un sistem de operare la altul, sau chiar între versiunile aceluiași sistem de operare.

X a fost creat în 1984 la MIT[5]. Versiunea curenta a protocolului este X11 și a apărut în 1987. În momentul de față proiectul X este condus de către X.Org Foundation și a ajuns la versiunea X11 Release 7.4.

---

[1] http://en.wikipedia.org/wiki/Code_page
[2] http://en.wikipedia.org/wiki/Comparison_of_Unicode_encodings
[3] http://en.wikipedia.org/wiki/Most_significant_bit
[4] http://en.wikipedia.org/wiki/UTF-8
[5] http://www.mit.edu

13.2.2 Arhitectura X Window System

X Window System are o arhitectură de tip client-server: mai mulți clienți X se conectează la un server X (figura 13.3). Serverul acceptă cereri de desenare a obiectelor grafice și trimite înapoi către clienți intrările (inputul) de la utilizator (preluate de la tastatură, mouse etc.).

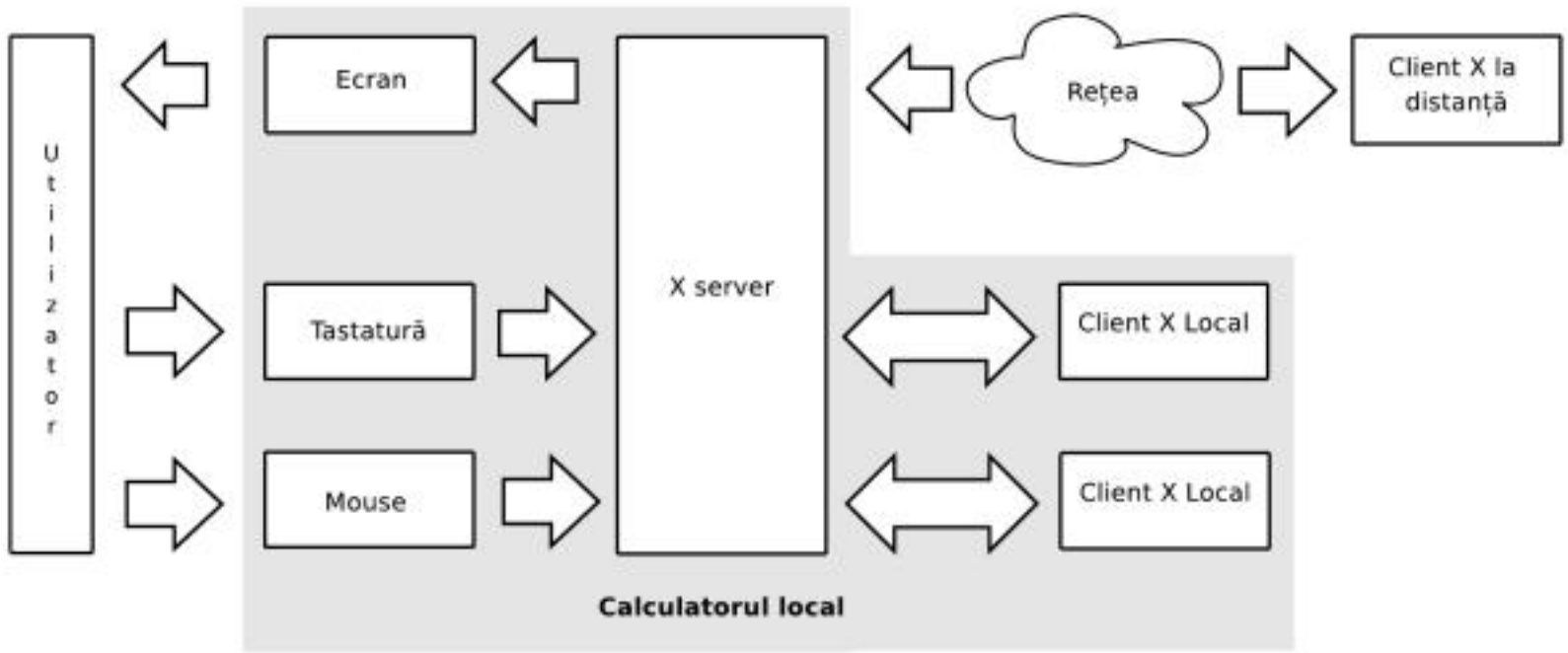


Figura 13.3: Arhitectura X Window System

Serverul X este responsabil de administrarea resurselor ce permit interacțiunea cu utilizatorul. Aceste resurse sunt: *ecranul*, *tastatura*, *mouse-ul*, *touchpad-ul*, *stylus-ul*, *joystick-ul* etc. Serverul X rulează pe calculatorul care interacționează cu utilizatorul și trimite intrările (inputul) primit de la acesta către clienți. Clienții procesează informația primită de la server și îi trimit acestuia înapoi informații pe care serverul le va afișa utilizatorului pe ecran. În prezent, cea mai utilizată implementare a server-ului X este oferită de proiectul X.Org[1].

Clienții X se conectează la un server X pentru a folosi resursele administrate de acesta. Ei cer serverului să afișeze pe ecran obiecte grafice și preiau de la acesta intrările (inputul) utilizatorului, precum apăsările de taste sau mișcările mouse-ului. Clienții pot rula local (pe același calculator cu serverul) sau pot rula pe un alt calculator aflat la distanță. Exemple de clienți X: un browser web, un client de e-mail, un program de mesagerie instant.

Protocolul X este protocolul utilizat în comunicația dintre clienți X și serverul X.

Comunicația dintre clienți și server se desfășoară diferit față de modelul general al comunicației client-server. În general clientul, ca localitate, se află pe calculatorul local iar serverul se află pe un calculator aflat la distanță. În cazul *X Window System* lucrurile stau invers. Serverul se află pe calculatorul local pentru a putea trimite și primi informații de la utilizator. Clienți se pot afla tot pe calculatorul local sau pe unul aflat la distanță.

În exemplul din figura 13.4, pe un calculator aflat la distanță rulează trei clienți X. Acest calculator poate să nu aibă conectat niciun monitor și nici mouse sau tastatură. Fiecare din cei trei clienți este conectat la un alt server X (fiecare server X fiind pe un calculator

[1] http://www.x.org/

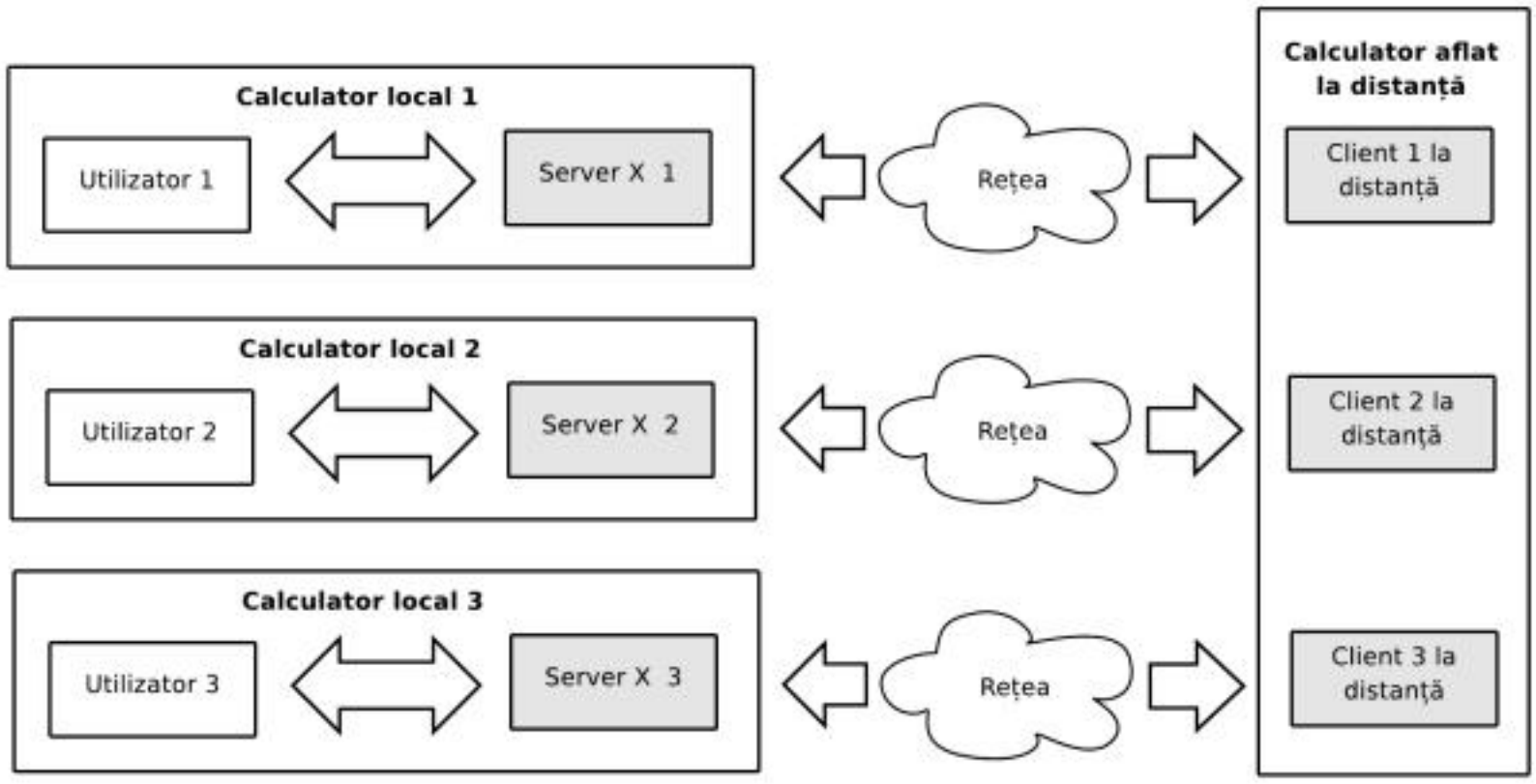


Figura 13.4: Exemplu de conectare client-server folosind protocolul X

diferit). Fiecare server X preia datele de la utilizatorul local (de exemplu prin intermediul tastaturii) și le transferă clientului, primind înapoi de la acesta informații care trebuie prezentate utilizatorului pe ecran. Toată procesarea informației de la utilizator este realizată de calculatorul aflat la distanță (acolo unde se execută clientul). Practic, în modelul de mai sus, calculatorul aflat la distanță se ocupa de procesarea datelor, calculatoarele 1, 2 și 3 funcționând pe post de terminale.

X nu conține nicio specificație legată de felul în care arată interfața cu utilizatorul (*Application User Interface*): modul în care arată butoanele, meniurile, ferestrele etc. Aceste specificații sunt implementate în schimb de către alte componente, precum *Window Manager*-ul (manager-ul de ferestre), *Desktop Environment*-ul (mediul desktop) sau de bibliotecile de componente grafice utilizate de aplicație aplicație.

Un window manager controlează amplasarea și modul în care arată ferestrele aplicațiilor. Acesta este un tip special de client X (se conectează la serverul X ca orice alt client) care controlează felul în care arată ceilalți clienți X.

> **Window manager**-ul face ca toți clienții să aibă caracteristici comune: bara de titlu a ferestrei, butoanele de minimize și maximize etc. El poate crea o interfață asemănătoare cu cea din Microsoft Windows sau cu cea din Mac OS X.

Window managerele pot oferi funcționalități de bază (precum `twm`[1], window managerul care vine împreună cu X) sau funcționalități complexe (cum ar fi Enlightenment ce se apropie ca funcționalitate de un desktop environment). Printre cele mai cunoscute window managere se numără: *Kwin*[2] este window managerul folosit de KDE, *Metacity*[3] cel folosit de GNOME și *Compiz*[4] este cel utilizat pentru desenarea interfeței 3D din proiectul cu același nume.

[1] http://en.wikipedia.org/wiki/Twm
[2] http://en.wikipedia.org/wiki/Kwin
[3] http://en.wikipedia.org/wiki/Metacity
[4] http://en.wikipedia.org/wiki/Compiz

În cazul în care această componentă lipsește, se vor putea crea ferestre noi dar nu vor avea bara de titlu. În plus, ele nu vor putea fi mutate pe ecran, nu vor putea fi închise cu `ALT+F4`, nu va funcționa `ALT+TAB`, nu se va putea muta o fereastră din fundal (*background*) în prim-plan (*foreground*). În schimb, conținutul ferestrelor va rămâne neschimbat.

Stilul în care arată ferestrele (aspectul lor vizual) este controlat de o componentă a window managerului numită **window decorator**. Acesta este responsabil cu desenarea titlului ferestrei, a butoanelor și a marginii acesteia.

Un **desktop environment** (mediu desktop) include un window manager, mai multe aplicații și o interfață consistentă din punct de vedere vizual.

Cele mai cunoscute medii desktop sunt KDE[1] (figura 13.5) și GNOME[2] (figura 13.6).

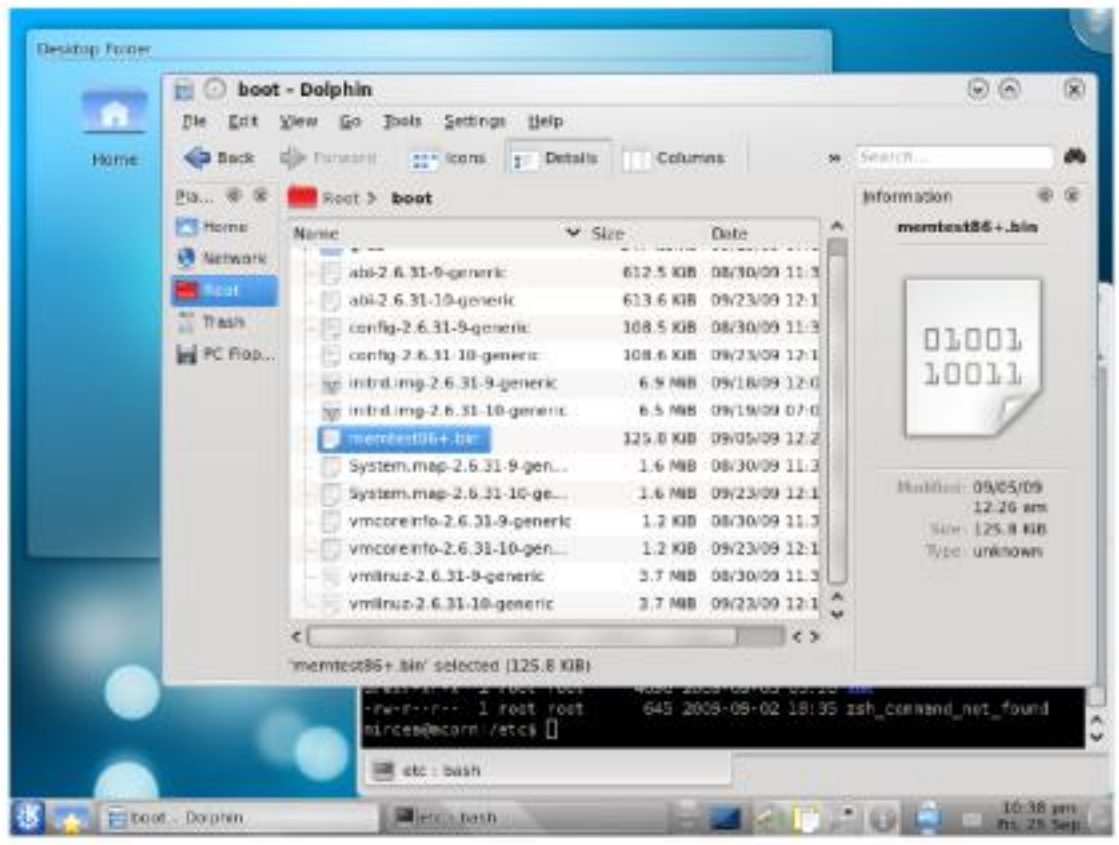


Figura 13.5: KDE

Un **display manager** este un program care permite pornirea unei sesiuni pe un server X de pe același calculator sau de pe unul aflat la distanță. Acesta este implementat ca un tip special de client X.

Display managerul afișează utilizatorului un ecran de autentificare (*login*) permițându-i acestuia să introducă un nume de utilizator și o parolă. După ce autentificarea s-a realizat cu succes o nouă sesiune este pornită. Exemplu de display managere: `kdm` (figura 13.7) folosit de KDE, `gdm` (figura 13.8) folosit de GNOME sau `xdm`, display managerul implicit din X Window System.

Atunci când display managerul rulează pe același calculator cu serverul X, el va porni serverul înainte de afișa fereastra de autentificare. Astfel display manager-ul are o funcție asemănătoare cu procesele `login` și `getty`(vezi secțiunea 6.5.3).

[1] http://en.wikipedia.org/wiki/KDE
[2] http://en.wikipedia.org/wiki/GNOME

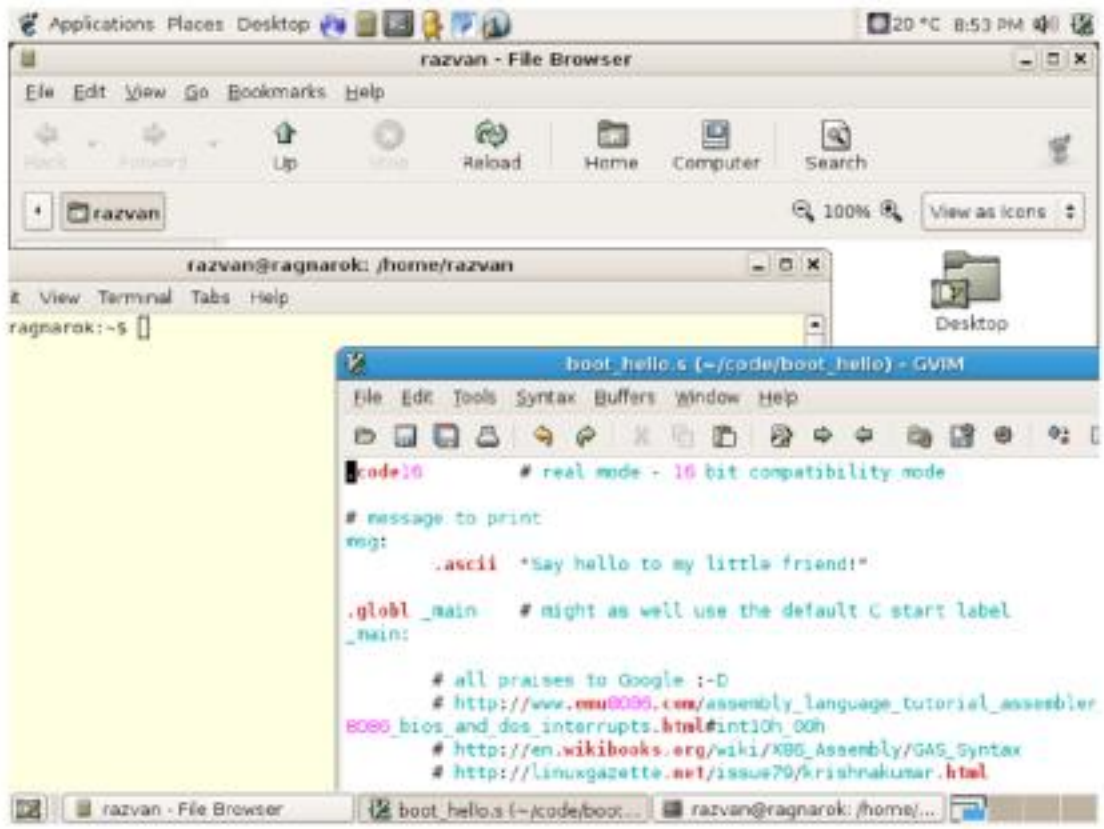


Figura 13.6: GNOME

Figura 13.7: Fereastra de login din Kubuntu (KDM)

13.3 Pornirea și oprirea interfeței grafice

În cazul în care interfața grafică nu pornește odată cu sistemul de operare, ea poate fi pornită separat. Pentru a realiza acest lucru există mai multe posibilități:

Folosind `xinit`

`xinit` este un program care pornește serverul X și un prim client care se conectează la el. Utilizatorul nu mai trebuie să se autentifice într-o fereastră de login deoarece el s-a autentificat în sistem înainte de a rula comandă **`xinit`**. Atunci când acest prim client își încheie activitatea, **`xinit`** va opri serverul X:

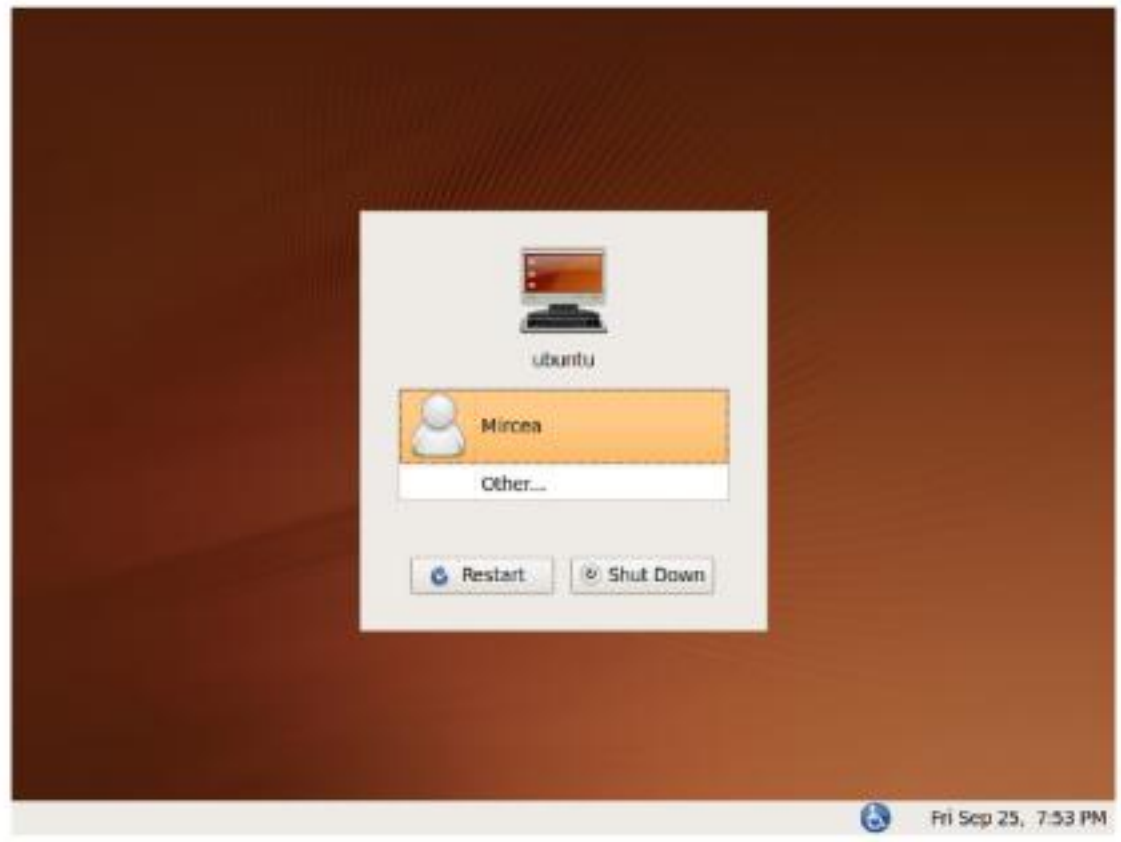


Figura 13.8: Fereastra de login din Ubuntu (GDM)

```
1 george@asgard:~$ man xinit
2 [...]
3 The xinit program is used to start the X Window System server and a first
   client program on systems that cannot start X directly from /etc/init or
   in environments that use multiple window systems.  When this first
  client exits, xinit will kill the X server and then terminate.
```

Folosind `startx`

`startx` este un front-end pentru **`xinit`** care oferă utilizatorului o interfață mai complexă:

```
1 george@asgard:~$ man startx
2 [...]
3 The startx script is a front end to xinit that provides a somewhat nicer
  user interface for running a single session of the X Window System. It is
   often run with no arguments.
```

Folosind un Display Manager

Atunci când un Display Manager este pornit pe calculatorul local, el va porni mai întâi un server X, urmând ca apoi să prezinte utilizatorului un ecran de autentificare. Dupa autentificare este pornit un server X în numele utilizatorului.

Pornirea unui Display Manager se poate face în mod asemănător serviciilor:

```
1 root@asgard:/home/george# service kdm start
```

Pentru **`gdm`** se procedează în mod asemănător.

Pentru oprirea interfeței grafice este suficientă oprirea display managerului. Acest lucru se poate realiza în felul următor:

```
1 root@asgard:/home/george# kill -9 *dm
```

Comanda anterioară oprește toate procesele al căror nume este oricât de lung și se termină cu șirul de caractere "`dm`".

O alta metodă de oprire a interfeței grafice este oprirea serviciului `kdm` (sau `gdm`) folosind paramerul `stop`:

```
root@asgard:/home/george# service kdm stop
```

Pentru oprirea și repornirea imediată a interfeței grafice este suficient să fie repornit display managerul folosind serviciile `kdm` (sau `gdm`) însă cu parametrul `restart`:

```
root@asgard:/home/george# service kdm restart
```

De asemenea oprirea și repornirea imediată a serverului X se poate realiza și prin combinația de taste `Ctrl+Alt+Backspace`.

13.4 Configurarea serverului X

Această secțiune se va concentra pe prezentarea modurilor de configurație temporară a server-ului X. KDE și GNOME vin cu aplicații pentru configurarea permanentă a server-ului X, salvând configurațiile în fișiere specifice Desktop Environment-ului.

În ultimele versiuni, necesitatea utilizării unui fișier de configurare a server-ului X a fost eliminată. Fișierul de configurare utilizat în versiunile anterioare se numea `/etc/X11/xorg.conf`.
Principalul dezavantaj al acestuia era faptul că trebuia repornit server-ul la fiecare modificare a fișierului. Fișierul mai poate fi utilizat în prezent pentru realizarea configurărilor permanente, dar, pentru simplitate în configurare, comenzile pentru configurarea temporară se pot salva în fișierul `~/.xprofile`.

Fișierul `/etc/X11/xorg.conf`, dacă este folosit, trebuie structurat pe secțiuni specifice fiecărui dispozitiv, după cum se poate observa și online[1] sau în documentația **man** (**`man xorg.conf`**).

13.4.1 Configurarea rezoluției

Versiunile recente ale server-ului X implementeaza extensia `Xrandr`, extensie care permite modificarea live (fără repornirea mediului grafic) a rezoluției, dimensiunii desktop-ului, orientării ecranelor etc. Acest lucru reprezintă un mare avantaj în cazul sistemelor la care se conecteaza monitoare externe (precum laptop-urile).

Extensia `Xrandr` poate fi utilizata folosind comanda **`xrandr`**. Câteva utilizări frecvente ale comenzii sunt prezentate mai jos:

- afișarea stării curente a sistemului grafic

```
mircea@acorn:~$ xrandr
Screen 0: minimum 320 x 200, current 1280 x 1024, maximum 1280 x
1280
VGA connected 1280x1024+0+0 (normal left inverted right x axis y
axis) 338mm x 270mm
   1280x1024      60.0*+   75.0     60.0*
   1152x864       75.0
```

[1] https://wiki.ubuntu.com/X/Config/Resolution

```
   1024x768        75.0     70.1     60.0
   832x624         74.6
   800x600         72.2     75.0     60.3     56.2
   640x480         75.0     72.8     66.7     59.9
   720x400         70.1
```

- activarea tuturor monitoarelor conectate la rezoluția lor default:

```
mircea@acorn:~$ xrandr --auto
```

- dezactivarea unui monitor (LCD-ul în acest caz):

```
mircea@acorn:~$ xrandr --output LVDS --off
```

- activarea unei anumite rezoluții (se poate preciza opțional și rata de refresh – pentru 75 Hz se folosește parametrul `--rate 75`)

```
mircea@acorn:~$ xrandr --output LVDS --mode 1024x768
```

13.4.2 Configurarea tastaturii

Printre cele mai importante configurări ale tastaturii se numără aranjamentul caracterelor pe taste (*keyboard layout*). Această configurație este importantă mai ales în condițiile în care un utilizator dorește să scrie caractere care nu se găsesc implicit pe tastatură.

În limba română s-au impus mai multe standarde de-a lungul timpului. Unele presupuneau înlocuirea completă a unor caractere de pe tastatură pentru includerea diacriticelor. În 2004 a fost decis un nou standard pentru aranjamentul caracterelor românești pe tastatură, cu următoarele specificații (pe scurt):

- caractere românești sunt disponibile ca aranjament secundar pentru tastatura US, apăsând tasta `AltGr` (tasta `Alt` din dreapta) – majoritatea tastaturilor din România au aranjamentul US, foarte puține fiind inscripționate cu diacritice românești în vechiul standard
- diacriticele `ă`, `î`, `ș`, `ț` sunt disponibile apăsând `AltGr` și aceleași taste dar fără virgulă (respectiv `a`, `i`, `s`, `t`)
- `â` este accesibil pe tasta `q`
- pentru a obține literele mari se apasă și tasta `Shift`

Astfel, pentru a obține caracterul `Ț` se apasă tastele: `AltGr+Shift+t`; pentru a obține caracterul `â` se apasă `AltGr+q`.

Pe lângă acestea, aranjamentul permite și obținerea caracterului €(euro) prin apăsarea combinației de taste `AltGr+e`.

O altă problemă în cazul aranjamentului de tastatură pentru limba română este modul în care se formează diacriticele `ș` și `ț`. Mulți utilizatori nu remarcă faptul că aceste două caractere (și literele mari echivalente) au două reprezentări posibile: *cu virgulă* (virgula este la o mică distanță de caracter) și *cu sedilă* (virgula este lipită de caracter). Standardul realizat de *Academia Română* menționează că diacriticele se construiesc folosind virgulă. O mare parte din implementările de aranjamente din sistemele de operare sunt greșite din acest punct de vedere.

Pentru a activa aranjamentul de tastatură pentru limba română conform standardului, în Linux se utilizează comanda:

```
1 mircea@acorn:~$ setxkbmap ro
```

Varianta *cu sedilă*, în caz că este necesară pentru compatibilitate, se poate activa folosind aceeași comandă dar adăugând și parametrul `cedilla`.

13.5 Configurarea sistemului din KDE

În Kubuntu, toate configurările importante ale sistemului sunt reunite într-un panou de comandă numit **System Settings**. Acest panou este asemănător ca funcționalitate cu **Control Panel** din Windows și poate fi accesat din meniul de KDE (aflat în stânga jos, în mod implicit).

13.5.1 System Settings

Pentru Kubuntu 9.10, fereastra de configurare este prezentată în figura 13.9.

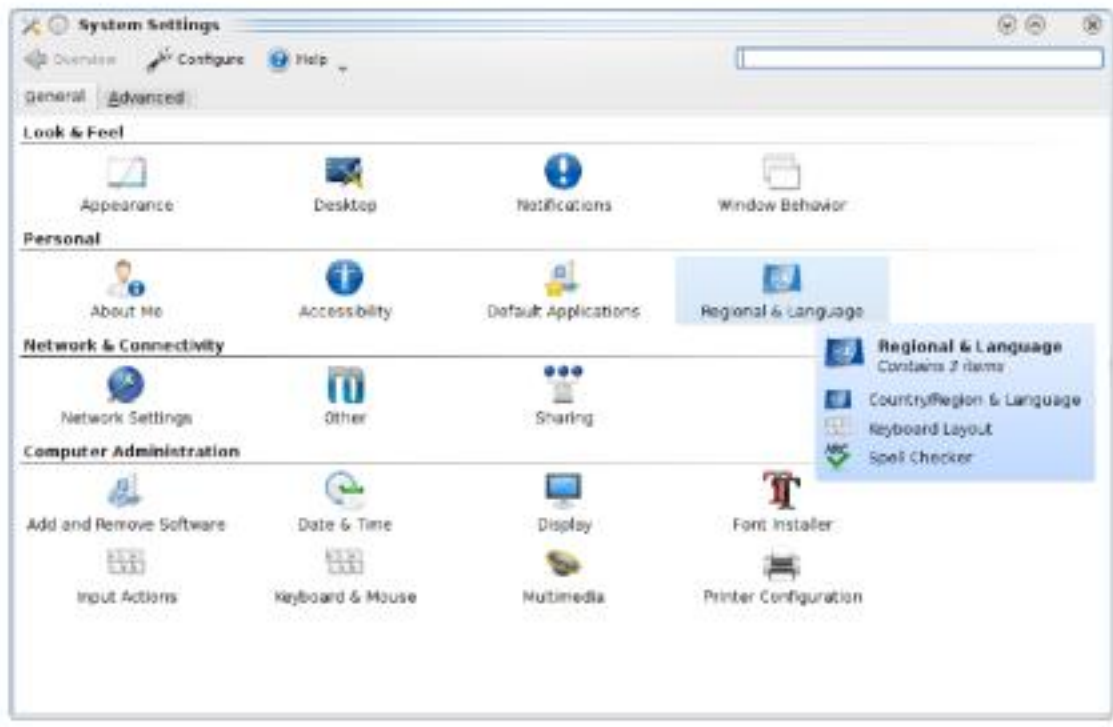


Figura 13.9: System Settings în Kubuntu 9.10

Ea oferă acces la principalele configurări desktop-ului și ale sistemului de operare: configurări legate de aspectul vizual, configurarea perifericelor, configurarea rețelei etc.

13.5.2 Schimbarea aspectului interfeței grafice

Appearance conține configurări legate de modul în care arată interfața grafică. Se poate alege schema de culori a ferestrelor și a butoanelor (*Colors*), fonturile folosite pentru afișarea textelor din interfața grafică (*Fonts*), icoanele folosite în sistem (*Icons*), stilul în care arată elementele vizuale(*Style*), felul în care arată bara de titlu a ferestrelor (*Window Decorations*), animația care arată stadiul curent al procesului de înărcare KDE (*Splash Screen*). Figura 13.10 prezintă un exemplu de configurări vizuale.

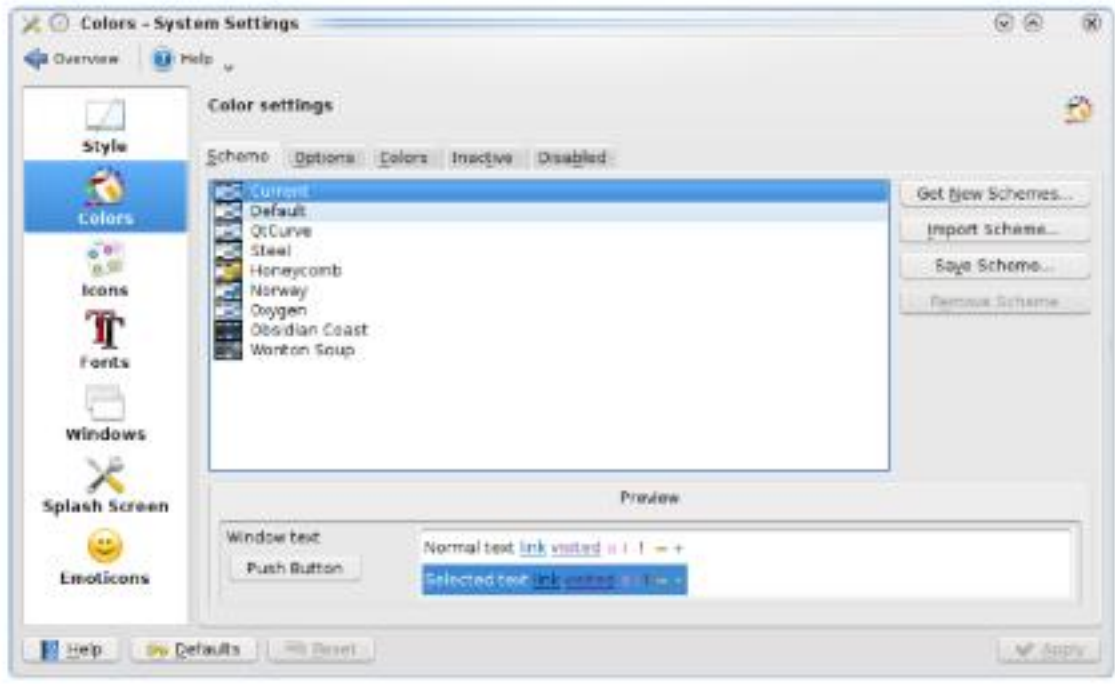


Figura 13.10: Alegerea schemei de culori

Desktop permite modificarea fundalului (*Background*), a screen saver-ului (*Screen Saver*) sau a numărului de desktop-uri (*Multiple Desktops*). În mod normal în Linux se folosesc două sau patru desktop-uri. Fiecare poate avea propria imagine de fundal și propriile programe deschise.

În general, aplicațiile sunt grupate de către utilizatori pe desktop-uri în funcție de utilitate. De exemplu, pe desktop-ul 1 se poate deschide un web browser, pe desktop-ul 2 un client de e-mail, pe desktop-ul 3 o temă de casă și pe desktop-ul 4 un program de mesagerie instant. Avantajul oferit de această abordare este acela că fiecare desktop devine mai liber, având mai puține programe deschise. De asemenea, acest mod de organizare permite mai ușor concentrarea atenției pe anumite activități.

Notifications conține configurări legate de sunetele asociate diferitelor acțiuni din sistemul de operare (*System Notifications*). De asemenea tot în această secțiune se pot defini și aplicațiile care pornesc atunci când un disc este inserat în unitatea optică (echivalentul opțiunii autorun din Windows) (*Storage Media Notifications*).

13.5.3 Configurări de bază ale sistemului de operare

About Me permite introducerea datelor personale ale utilizatorului și specificarea căii către directoarele sale personale: directorul *home*, directorul unde se află fișierele de pe *Desktop* și directorul `Autostart`. În directorul de `Autostart` se pot pune scripturi care vor fi rulate la pornirea KDE.

Accesibility oferă acces la diferire configurări pentru persoanele cu dizabilități.

Default Applications permite specificarea aplicațiilor folosite la deschiderea diferitelor tipuri de fișiere.

Regional & Language conține configurări legate de regiunea în care se găsește utilizatorul (de exemplu modul în care sunt afișate numerele, separatorul pentru partea zecimală a numerelor, simbolul pentru moneda utilizată, tipul de calendar folosit, formatul orei și al datei). Tot în această secțiune se poate alege tipul de tastatură (de exemplu cu 104 sau 105 taste) și aranjamentul caracterelor pe taste. Figura 13.11

prezintă configurația pentru tastatura în limba română.

Figura 13.11: Tastatura în limba română

13.5.4 Configurări administrative

Date & Time permite configurarea datei și a orei sistemului. Tot aici se poate alege și fusul orar corespunzător zonei în care se găsește utilizatorul.

Keyboard & Mouse conține configurări legate de tastatură și mouse. Una din cele mai importante opțiuni prezente aici este cea legată de starea `NumLock` la inițializarea KDE. Se poate alege ca `NumLock` să fie activat, dezactivat sau să își păstreze starea anterioară. În cadrul subsecțiunii *Mouse* se poate alege tema cursorului de la mouse.

Tot în această secțiune se pot configura combinațiile de taste (*shortcut*) pentru diferite funcții din KDE. Cele mai utilizate combinații (cu valorile lor implicite) sunt prezentate în tabelul 13.1.

Display permite, printre altele, alegerea uneia dintre rezoluțiile de ecran disponibile și a parametrilor de culoare (gamma, luminozitate, contrast).

Multimedia conține configurări ale plăcii audio.

13.5.5 Configurarea rețelei

În *System Settings*, la **Network Settings** se pot face configurări ale rețelei.

În *Network Connections* > *Wired* apare lista conexiunilor configurate (figura 13.12). Alegând una din aceste conexiuni se pot configura parametrii ei [referință la parametrii de rețea], precum în figura 13.13.

13.5.6 Managementul utilizatorilor

Configurarea utilizatorilor se realizează folosind aplicația KDE User Manager (**`kuser`**) – figura 13.14.

Tabelul 13.1: Combinații de taste uzuale în KDE

Combinația de taste	**Acțiune**
`F1`	Deschide fereastra de ajutor
`Alt+F2`	Deschide fereastra de Run de unde se poate porni o aplicație specificând numele executabilului
`Ctrl+Esc`	Pornește aplicația *System Activity* în care este afișată o listă a proceselor active
`Ctrl+Fn`	Face trecerea la desktop-ul `n` (`n` este un număr între 1 și 4)
`Alt+Tab` sau `Alt+Shift+Tab`	Face trecerea de la o aplicație la alta. Folosind tasta `Shift` se ciclează aplicațiile în ordine inversă
`Alt+F4`	Închide fereastra activă
`Ctrl+Alt+Esc`	Această combinație permite terminarea unei aplicații în mod forțat. După ce combinația de taste a fost efectuată, cursorul mouse-ului se va transforma (cel mai uzual într-un `X`) și fereastra pe care se va realiza click va fi terminată în mod forțat. Funcționarea aplicației este similară comenzii **`kill`**

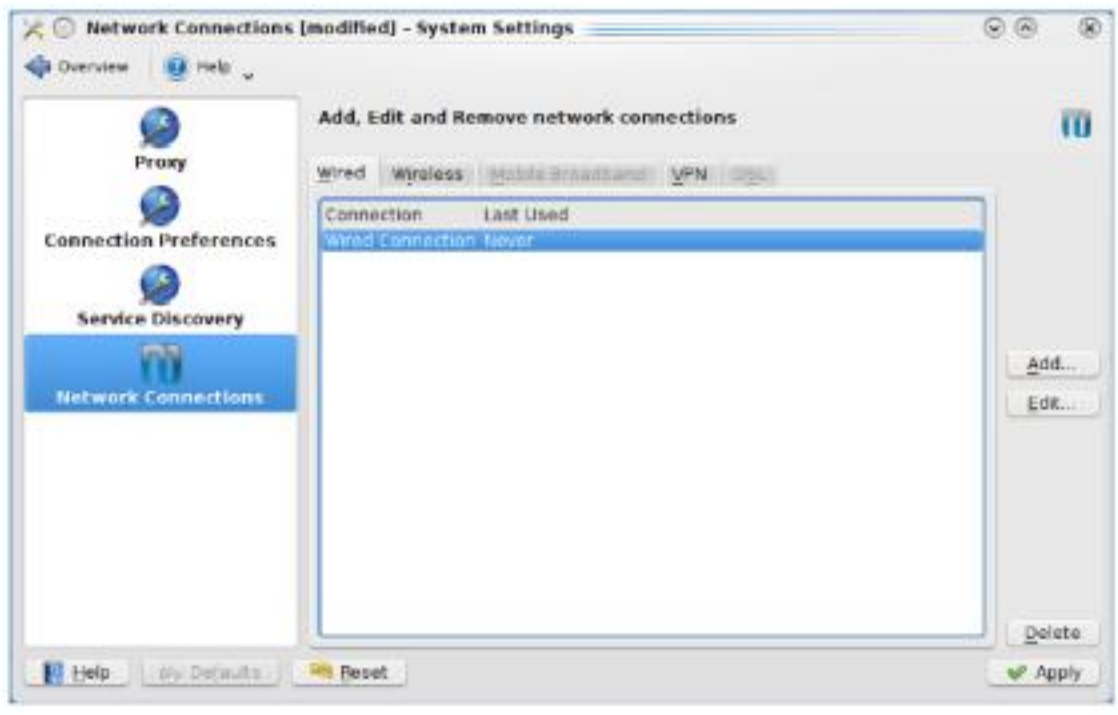


Figura 13.12: Configurarea conexiunilor la rețea

Aici se pot adăuga, șterge sau modifica utilizatorii din sistem. Se pot modifica numele de login, numele real al utilizatorului, grupurile din care face parte etc. Pentru a modifica datele unui utilizator se selectează utilizatorul dorit și se apasă butonul *Modify*, fereastra afișată fiind prezentată în figura 13.15.

13.5.7 Monitorizarea sistemului și managementul proceselor

Aplicația KDE pentru monitorizarea sistemului este **System Monitor** (comanda **`ksysguard`**). Aceasta vine implicit împreună cu Kubuntu.

Aplicația are două tab-uri în configurația implicită: unul pentru managementul și monitorizarea proceselor (figura 13.16) și unul pentru monitorizarea sistemului

Figura 13.13: Configurarea parametrilor de rețea de bază

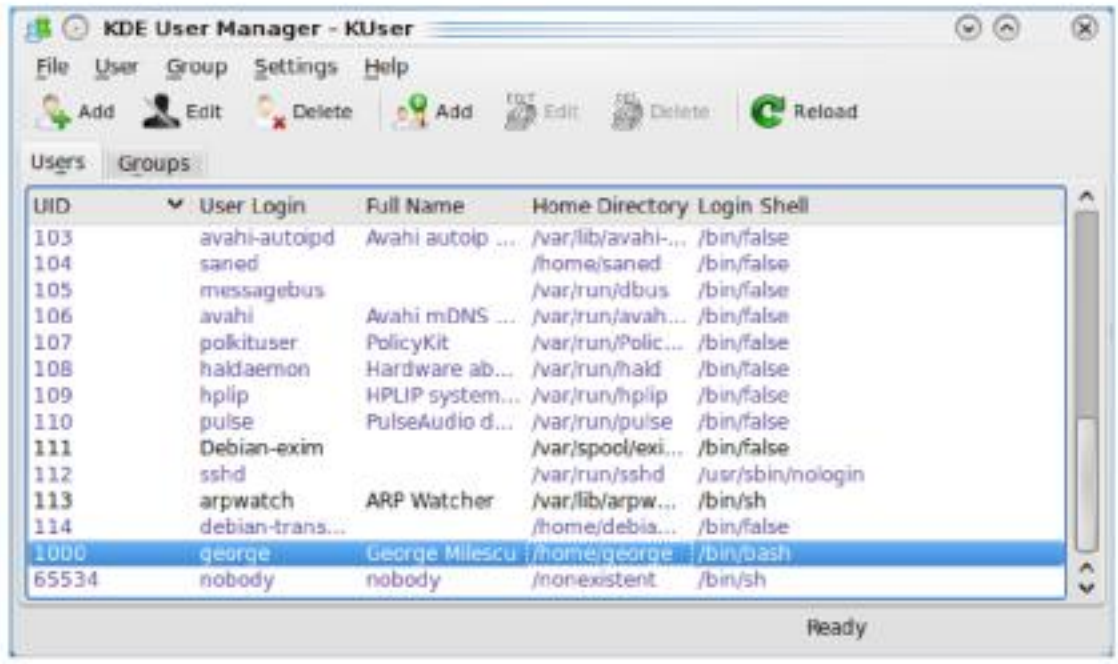


Figura 13.14: Administararea utilizatorilor în KDE User Manager

(figura 13.17). Dacă se dorește, lista de procese poate fi afișată sub formă de arbore.

De asemenea, în partea de sus a ferestrei se află un câmp *Search* se permite căutarea unui proces în listă după numele lui. Un proces poate fi oprit în mod forțat dacă este selectat din listă și este apăsat butonul *Kill*.

Pentru monitorizarea sistemului, aplicația pune la dispoziție o colecție de senzori ale căror valori pot fi grupate în tab-uri personalizate, precum se poate observa în figura 13.18.

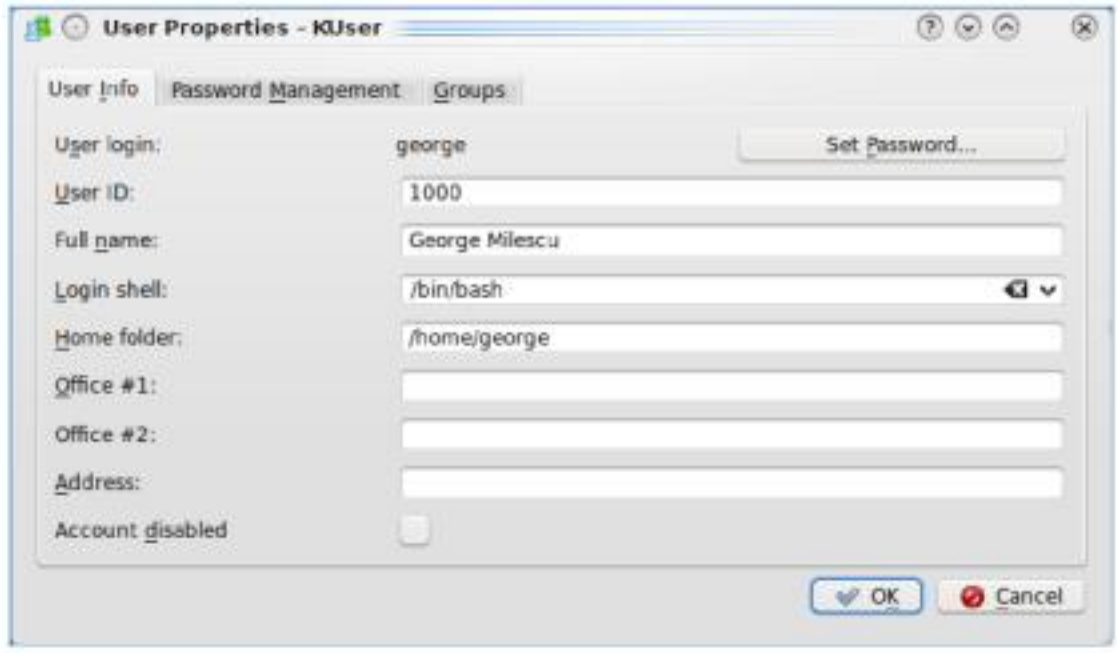


Figura 13.15: Modificarea datelor unui utilizator

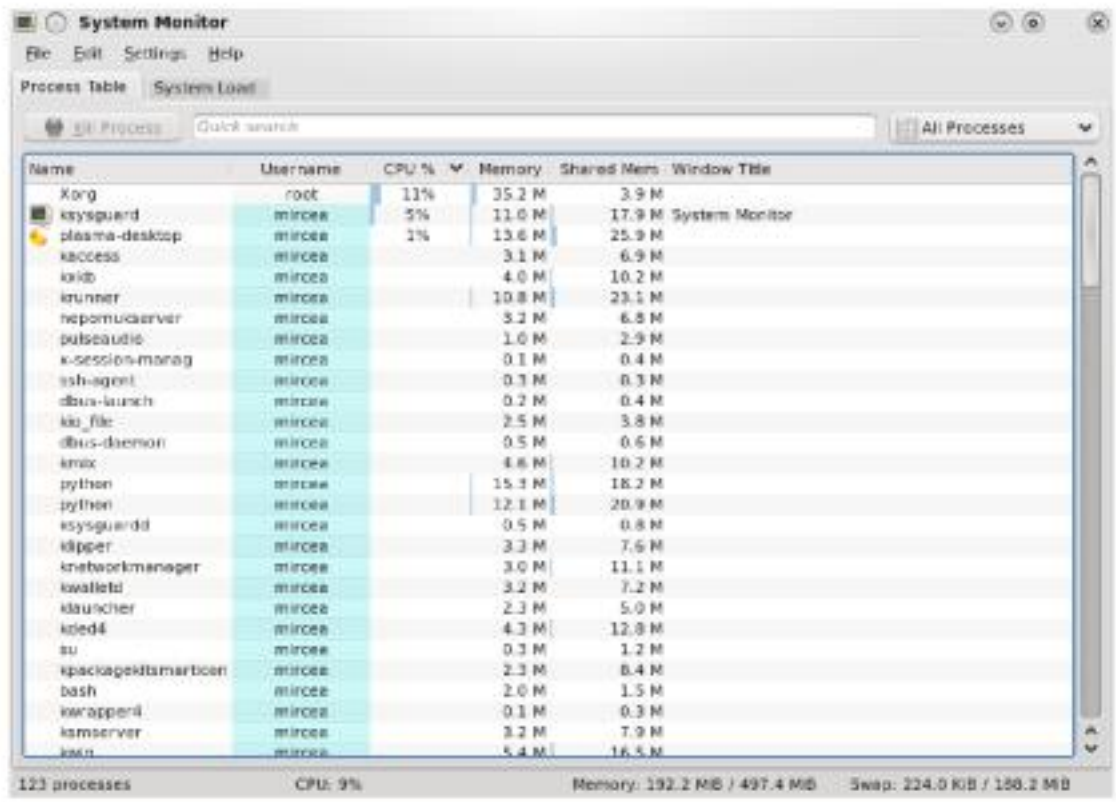


Figura 13.16: System Monitor – lista de procese

13.5.8 Gestiunea pachetelor

Există mai multe utilitare pentru a realiza gestiunea pachetelor folosind interfața grafică. În Kubuntu, aplicația utilizată pentru managementul pachetelor se numește *KPackageKit*. Figura 13.19 prezintă interfața *KPackageKit* pentru realizarea actualizărilor.

Aplicația permite gestiunea repository-urilor, actualizarea bazei de date cu pachete, căutarea de pachete după nume, instalarea de pachete noi, dezinstalarea de pachete și actualizarea tuturor pachetelor din sistem.

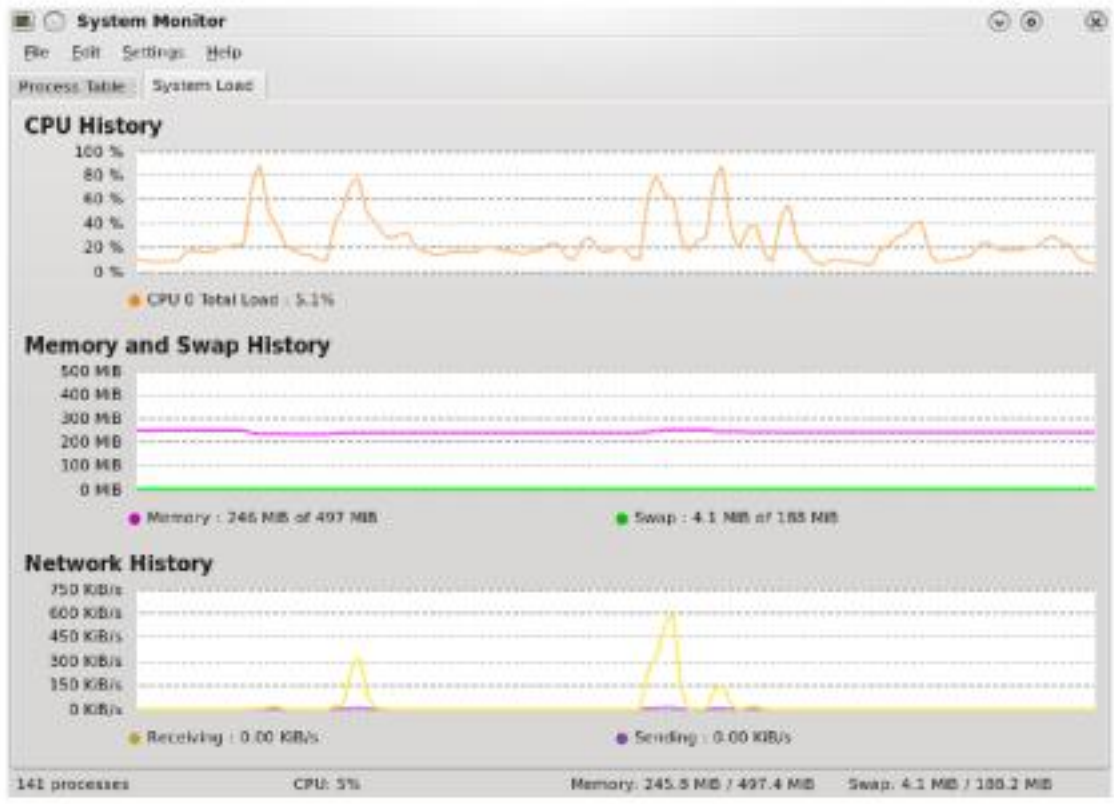


Figura 13.17: System Monitor – monitorizarea sistemului

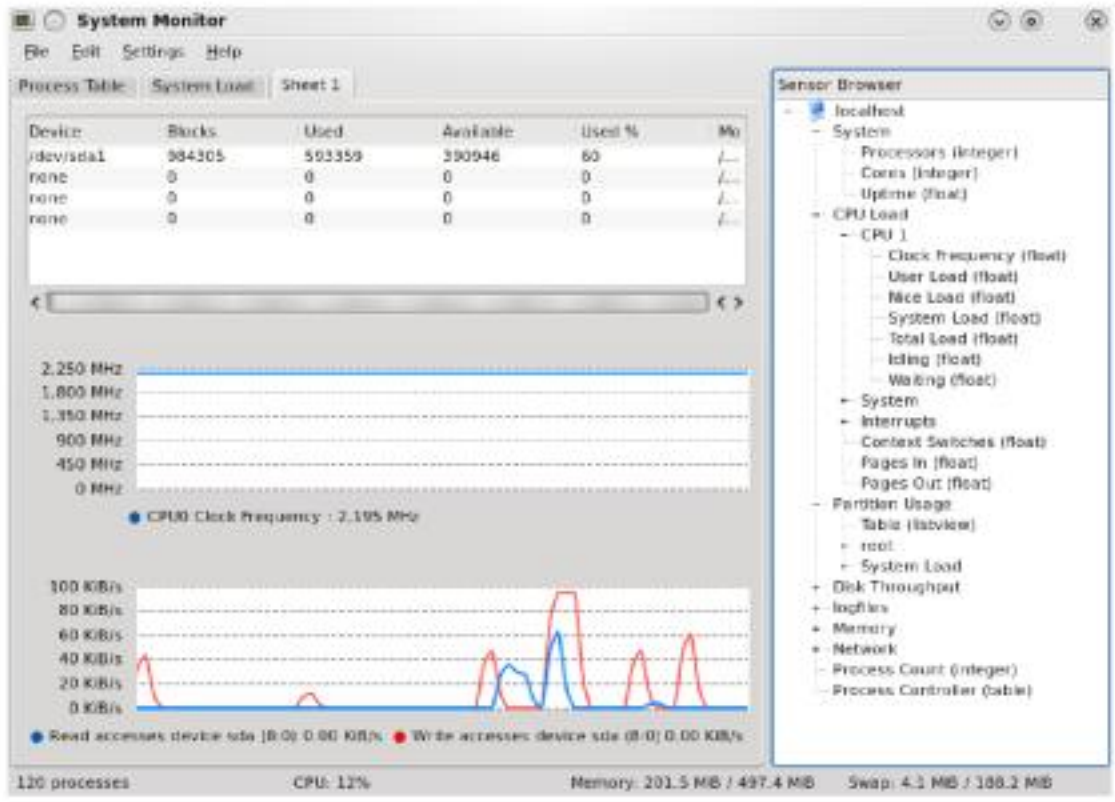


Figura 13.18: System Monitor – senzori care pot fi folosiți în tab-uri personalizate

13.6 Servicii desktop

13.6.1 Pornirea facilă a aplicațiilor

KDE versiunea 4 a introdus în KDE un nou serviciu de pornire a aplicațiilor. El înlocuiește vechea fereastră de Run (asemănătoare cu cea din Windows) cu o fereastră în care utilizatorul este ajutat dinamic să identifice aplicația pe care dorește să o pornească. Accesul la aceasta fereastra de execuție se face folosind combinația de taste `Alt+F2` – aceeași combinație ca în KDE3 și în Gnome.

În urma introducerii câtorva litere din numele sau descrierea aplicației (figura 13.20), fereastra se va completa cu aplicațiile posibile. Aceeași fereastră poate fi folosită și

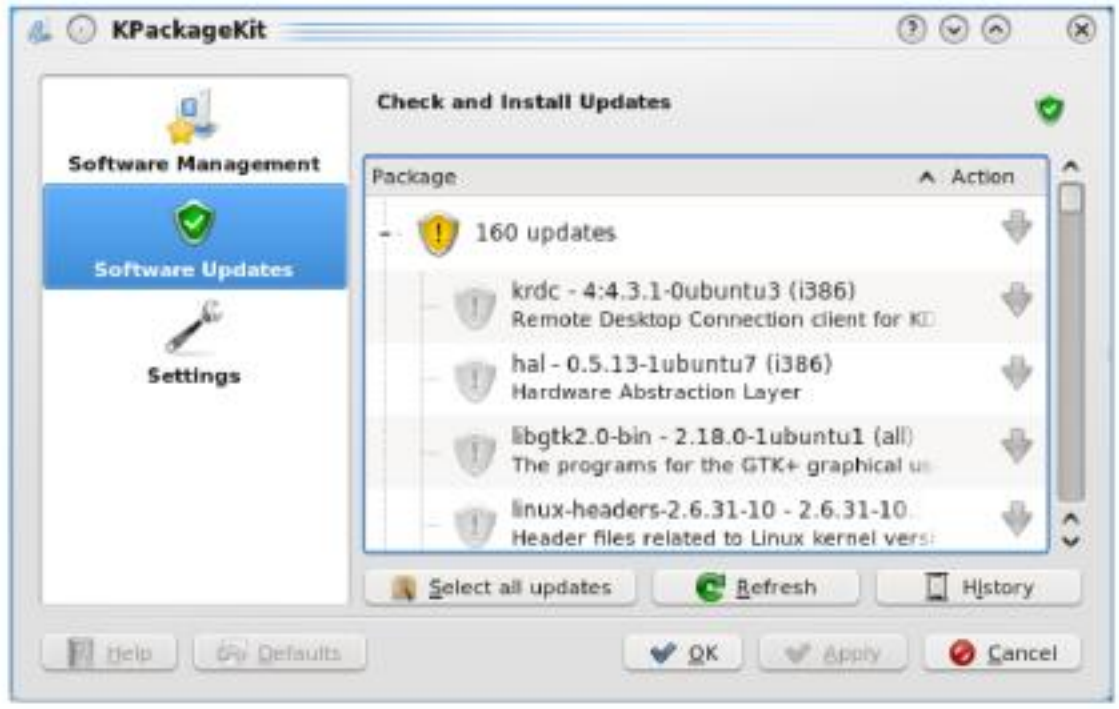


Figura 13.19: KPackageKit – aplicația pentru managementul pachetelor

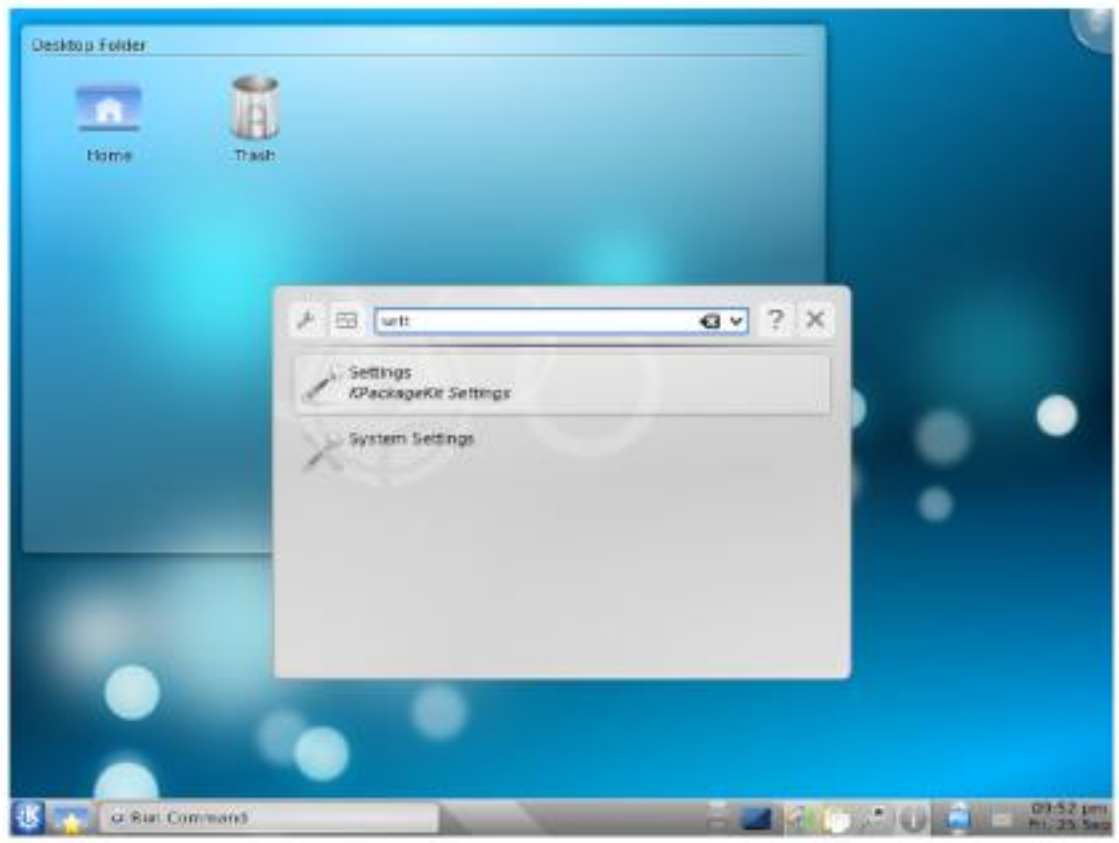


Figura 13.20: Fereastra de execuție a aplicațiilor

pentru realizarea de calcule simple – doar se introduce calculul dorit urmat de semnul egal (spre exemplu: `.19 * 2000 =`) și rezultatul apare în listă.

13.6.2 Căutarea fișierelor în sistem

În prezent, pentru ca utilizatorul să poată găsi date pe calculatorul propriu, fiecare sistem de operare modern oferă un serviciu de căutare a fișierelor. În general, acest serviciu se bazează pe un daemon care indexează permanent fișierele din sistem. La indexare, pe lângă numele și locația fișierelor, daemon-ul de indexare poate colecta atât conținut cât și metadate ale fișierului (spre exemplu, în cazul fișierelor `MP3`, se indexează și tag-urile fișierului precum artistul, numele melodiei etc.).

Serviciile de căutare din sistemele de operare pe cele mai cunoscute sunt următoarele:

- Windows – *Windows Search*[1]
- Linux – Strigi[2] (KDE), Beagle[3] (GNOME)
- Mac OS X – *Spotlight*[4]

Utilizatorul poate instala și soluții *third-party* pentru realizarea căutărilor în sistem. Cea mai cunoscută soluție pentru cautare pe desktop oferită ca aplicație separată este *Google Desktop Search*[5].

13.6.3 Notificări

Sistemele de operare moderne pun la dispoziția aplicațiilor servicii de notificare a utilizatorului. Astfel, notificările au un aspect consistent, efortul pentru dezvoltarea aplicațiilor scade și crește nivelul de utilizabilitate. În Linux există suport pentru notificări generice prin intermediul bibliotecii `libnotify`.

Pe lângă sistemele de notificare puse la dispoziție de desktop, majoritatea interfețelor grafice pun la dispoziție o zonă în dreptul ceasului pentru icoanelor aplicațiilor care doresc să notifice utilizatorul (pe bara de aplicații sau pe bara cu meniul desktop-ului). Această zonă mai este numită *tray* (sau *system tray*).

13.7 Aplicații KDE vs. GNOME

Alegerea între KDE și GNOME aparține fiecărui utilizator și se poate baza pe mai multe criterii, precum aspectul vizual și nivelul de funcționalitate dorit. Tabelul 13.2 prezintă comparativ aplicațiile cele mai uzuale de KDE și GNOME.

Aplicațiile prezentate nu reprezintă restricții de utilizare. Ele se pot executa atât pe și KDE cât și pe GNOME dacă sunt instalate bibliotecile necesare[6].

13.8 Studii de caz

13.8.1 Configurarea X peste SSH

Așa cum a fost prezentat și la începutul capitolului, arhitectura **X Window System** permite comunicarea prin rețea între clienții X și serverul X. Această comunicare se poate realiza folosind mai multe protocoale, printre care și SSH. SSH este un protocol care permite comunicarea între două calculatoare utilizând un canal securizat (secțiunea 9.2.2). SSH poate astfel securiza mesajele dintre clienții și serverul X

[1] http://www.microsoft.com/windows/products/winfamily/desktopsearch/default.mspx
[2] http://strigi.sourceforge.net/
[3] http://beagle-project.org/
[4] http://www.apple.com/macosx/what-is-macosx/spotlight.html
[5] http://desktop.google.com/
[6] bibliotecile sunt instalate automat la instalarea aplicațiilor

Tabelul 13.2: Aplicații KDE și GNOME

Aplicație	KDE	GNOME
Configurarea Sistemului	System Setings	Meniul System
Gestiunea pachetelor	KPackageKit	Synaptic Packet Manger
Monitorizarea sistemului	System Monitor (KSysGuard)	System Monitor
File Manager	Konqueror, Dolphin	Nautilus
Editor de text	Kate	Gedit
Emulator de consolă	Konsole	gnome-terminal
Calculator	Kcalc	Gcalctool
Video Player	Kaffeine	Totem
Audio Player	Amarok	Rhythmbox
Înscriptor de CD/DVD	K3B	Brasero
Viewer de imagini	Gwenview	Eye of GNOME
Viewer PDF	Okular	Evince
Browser Web	Konqueror	Epiphany

(figura 13.21). Opțiunea SSH care permite acest lucru poate se numește *X11 Forwarding*.

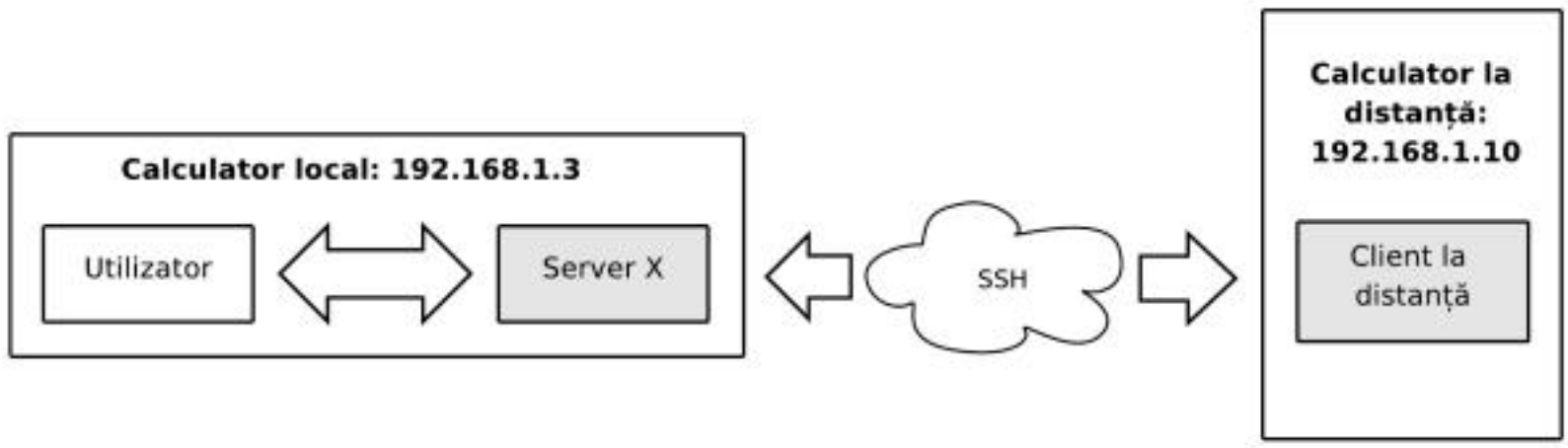


Figura 13.21: Conexiunea client – server X peste SSH

Pentru a putea rula un client X aflat la distanță peste SSH, primul pas este configurarea protocolului SSH în acest sens (activarea *X11 Forwarding*). Cea mai simplă metodă este specificarea parametrului `-X` atunci când se realizează conexiunea SSH de pe calculatorul utilizatorului (unde este server-ul X) pe calculatorul unde se va executa clientul X.

Comanda cu care se realizează conexiunea SSH pentru a permite încapsularea protocolului X este:

```
1 george@asgard:~$ ssh george@192.168.1.10 -X
```

După ce conexiunea SSH a fost stabilită, se poate porni un client X. De exemplu, se poate porni *KCalc*:

```
1 george@acasa:~$ kcalc
```

După cum se poate observa în figura 13.22, bara de titlu a ferestrei *KCalc* specifică faptul că această fereastră corespunde unui client care nu se găsește pe calculatorul local (`<@acasa>`).

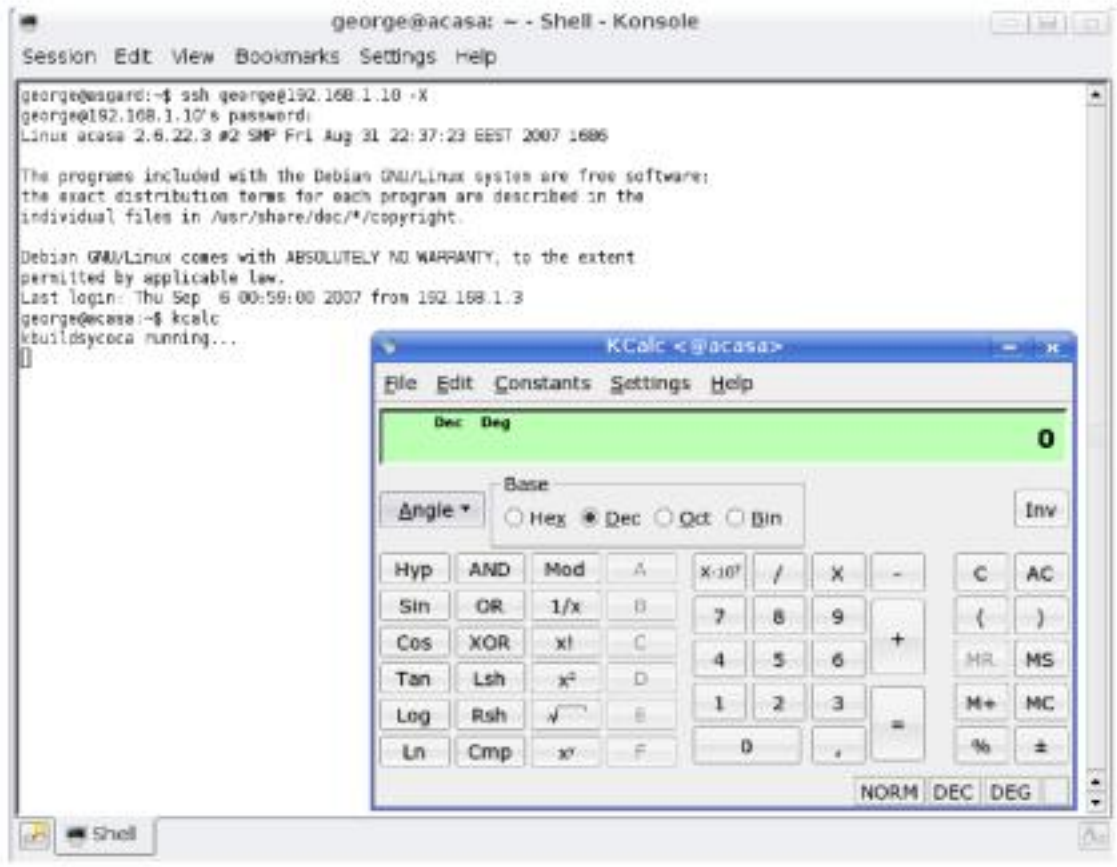


Figura 13.22: Pornirea unei aplicații X peste SSH

Folosind X peste SSH se poate executa și un întreg *Desktop Environment*, nu doar programe individuale. Acest lucru nu este însă uzual datorită alternativelor mai bune existente în acest sens (**vnc**, **freenx** etc.). Conexiunea SSH necesită criptare și decriptare – acest lucru presupune consumul unor resurse hardware suplimentare.

Pentru a putea rula X peste SSH folosind pe calculatorul local Windows (server X) și Linux pentru clienții X, trebuie ca pe calculatorul local să fie instalat *Cygwin* (sau un alt server X instalabil pe Windows).

13.8.2 Instalarea și configurarea VNC

VNC (*Virtual Network Computing*) este unul dintre cele mai folosite sisteme de vizualizat desktop-ul unui calculator aflat la distanță. VNC se bazează pe protocolul *RFB* (*Remote Frame Bufer*) și de aceea poate fi folosit între sisteme de operare diferite (Windows, Linux, Mac OS X).

VNC funcționează pe sistemul client – server. Pe calculatorul aflat la distanță (al cărui desktop va fi vizualizat) rulează un server de VNC la care se conectează un client aflat pe calculatorul local. La un server se pot conecta mai mulți clienți simultan. Un client poate să vizualizeze desktop-ul calculatorului pe care rulează serverul VNC sau, dacă este permis din configurarea server-ului, poate să și interacționeze cu el (de exemplu schimbând poziția cursorului mouse-ului). Figura 13.23 prezintă o astfel de sesiune VNC.

Serverul de VNC preia de câteva ori pe secundă imaginea care se afișează pe ecran și o trimite tuturor clienților. În cazul în care clienții îi cer acest lucru, serverul poate accepta de la aceștia cereri de mișcare a cursorului mouse-ului, de efectuare de click-uri de mouse sau de introducere de caractere. În acest fel, utilizatorul poate prelua controlul calculatorului pe care rulează serverul de VNC.

Figura 13.23: Conectarea prin VNC folosind un client Windows și un server Linux

Folosind VNC este posibil ca mai mulți utilizatori să aibă simultan controlul unui calculator. De cele mai multe ori, înainte de a putea accesa un server de VNC, utilizatorul trebuie să se autentifice. Acest lucru se realizează pe baza unui nume de utilizator și a unei parole.

Atât în Windows cât și în Linux există mai multe programe, atât server cât și client, care implementează VNC. Pentru Linux, unul dintre cele mai simple servere VNC este **`x11vnc`**. Odată instalat, acest server este pornit prin apelarea comenzii **`x11vnc`**:

```
1  george@acasa:~$ x11vnc -usepw
2  10/09/2007 22:48:59 -usepw: found /home/george/.vnc/passwd
3
4  [...]
5  10/09/2007 22:48:59 *** Trying ":0" in 4 seconds.  Press Ctrl-C to abort.
6
7  10/09/2007 22:48:59 *** 1 2 3 4
8  10/09/2007 22:49:03 *** XOpenDisplay of ":0" successful.
9
10 10/09/2007 22:49:03
11 10/09/2007 22:49:03 Using X display :0
12 10/09/2007 22:49:03
13 10/09/2007 22:49:03 X display :0.0 is 32bpp depth=24 true color
14
15 FrameBuffer Info:
16  width:              1280
17  height:             1024
18  scaled_width:       1280
19  scaled_height:      1024
20
21 [...]
22
23 The VNC desktop is:      acasa:0
24 PORT=5900
```

Comanda trebuie apelată de către utilizatorul care este autentificat în interfața grafică. Implicit, după ce ultimul client se deconectează, serverul se va opri.

Pentru a securiza accesul clienților pe baza unei parole, **`x11vnc`** trebuie pornit cu opțiunea `-usepw`. În acest caz serverul va verifica dacă există fișierul

`~/.vnc/passwd`. În cazul în care fișierul există, din el va fi citită parola ce va fi cerută clienților. Dacă fișierul nu există, utilizatorului i se va cere o parolă ce va fi stocată în `~/.vnc/passwd` și va fi folosită la următoarele instanțe ale serverului.

Există mai mulți clienți VNC în Linux: **`xvnc4viewer`**, **`xvncviewer`** etc.

Şi în Windows există servere cât clienți VNC. Unele programe sunt gratuite (ca de exemplu **tightVNC**), altele trebuie cumpărate (ca de exemplu **RealVNC**, ce dispune și de o versiune gratuită dar limitată ca opțiuni).

O alternativă la VNC este **FreeNX**. Acest program trimite datele între client și server peste SSH, însă comprimă datele înainte de a le trimite. În acest fel reușește să obțină performanțe superioare VNC-ului ca timp de răspuns și ca lățime de bandă consumată.

13.8.3 Remote Desktop Connection

RDC (*Remote Desktop Connection*) este o aplicație care funcționează în mod similar cu VNC și care permite unui utilizator să acceseze de la distanță desktop-ul unui calculator. Această aplicație este disponibilă doar împreună cu *Windows XP Professional* sau cu *Windows Vista Business*, *Ultimate* sau *Enterprise*.

În comparație cu VNC, RDC are avantajul că este instalat odată cu instalarea Windows. Dezavantajul RDC este acela că nu permite accesarea desktop-ului simultan local și de la distanță. Cu alte cuvinte, atunci când un utilizator este autentificat local pe calculator și interacționează direct cu sistemul de operare, nu este posibil ca altcineva să fie autentificat prin RDC și să vadă acțiunile primului utilizator. În momentul în care se încearcă autentificarea prin RDC, utilizatorul local este deconectat. Atunci când un utilizator aflat la distanță este autentificat prin RDC și un utilizator local încearcă și el să se autentifice, utilizatorul aflat la distanță este deconectat.

Pentru utilizarea RDC, acesta trebuie activat pe calculatorul *server* (calculatorul care va fi controlat de la distanță). Pentru a realiza acest lucru pe Windows XP se accesează *System Properties* (click dreapta pe *My Computer*, apoi *Properties*), după care din tabul *Remote* se bifează *Allow users to connect remotely to this computer* (figura 13.24).

Este posibil să fie necesare configurări ale programelor de tip firewall (vezi secțiunea 10.5.1) existente pe cele două calculatoare între care se dorește realizarea legăturii.

Pentru a realiza conexiunea, de pe calculatorul local trebuie pornit clientul de RDC (figura 13.25). Acesta se accesează prin *Start* > *All Programs* > *Accessories* > *Communications* > *Remote Desktop Connection* sau *Start* > *All Programs* > *Accessories* > *Remote Desktop Connection*.

Pentru conectarea la calculatorul aflat la distanță este necesară adresa IP a acestuia. Folosind butonul *Options* se pot configura parametrii conexiunii RDC (figura 13.26).

După ce conexiunea a fost inițializată, utilizatorul trebuie să se autentifice (figura 13.27).

După ce autentificare este realizată cu succes, desktop-ul calculatorului afla la distanță poate fi accesat, asemănător unei conexiuni VNC.

Figura 13.24: Activarea RDC

Figura 13.25: Fereastra clientului de RDC

Cuvinte cheie

- aplicații grafice
- client X
- desktop environment: KDE, GNOME
- diacritice
- display manager: **`gdm`**, **`xdm`**, **`kdm`**
- font
- grafică raster
- grafică vectorială
- GUI
- keyboard layout
- protocolul X
- rezoluție
- RDC
- server X
- servicii desktop
- Unicode
- UTF-8
- VNC
- window manager: Kwin, Metacity, Compiz
- X Forwarding
- X Window System
- X.Org
- **`xinit`**, **`startx`**
- Xrandr

Figura 13.26: Configurarea opțiunilor conexiunii RDC

Figura 13.27: Autentificarea RDC

Întrebări

1. Care din următoarele formate de fișiere NU este un format vectorial?
   - ❑ PDF
   - ❑ True Type font
   - ❑ PNG
   - ❑ SVG
2. Care dintre următoarele este un standard pentru descrierea modul de codificare a caracterelor intr-un fișier?

- ❑ True Type
- ❑ Unicode
- ❑ UTF-8
- ❑ RGB

3. Un Display Manager are rolul de a:
   - ❑ controla felul în care arată ferestrele
   - ❑ permite unui utilizator să se autentifice în interfața grafcă
   - ❑ controla deschiderea și închiderea ferestrelor
   - ❑ gestiona conținutul ferestrelor
4. Care dintre următoarele perechi NU conține elemente cu același rol?
   - ❑ KDE și GNOME
   - ❑ **kdm** și **gdm**
   - ❑ Kwin și Metacity
   - ❑ Xorg și bitmap
5. Utilizatorul Dorel lucrază pe un calculator cu interfață grafică instalată, dar care NU este pornită. El încearcă să pornească interfața folosind comanda următoare dar nu reușește. De ce?

```
1 dorel@home:~$ service kdm start
```

   - ❑ scriptul `kdm` nu porneste interfața grafică
   - ❑ Dorel nu are dreptul să pornească serviciul `kdm`
   - ❑ Dorel trebuia să ruleze scriptul **`startx`** înainte de a rula scriptul `kdm`
   - ❑ Dorel trebuia să ruleze scriptul **`startx`** după ce a rulat scriptul `kdm`
6. Care dintre urmatoarele NU este o metoda de control de la distanță a unui calculator?
   - ❑ X peste SSH
   - ❑ VNC
   - ❑ Spotlight
   - ❑ Remote Desktop Connection
7. Care dintre următoarele NU este un Window Manager?
   - ❑ KDE
   - ❑ Compiz
   - ❑ Kwin
   - ❑ Metacity

8. Care dintre următoarele aplicații este utilizată în mediul grafic pentru monitorizarea ocupării resurselor hardware?
   - ❑ top
   - ❑ System Information (**`ksysguard`**)
   - ❑ htop
   - ❑ free
9. Un Window Manager controlează felul în care arată ferestrele aplicațiilor. Un client X preia intrărea (input-ul) direct de la utilizator.
   - ❑ adevărat, adevărat
   - ❑ adevărat, fals
   - ❑ fals, adevărat
   - ❑ fals, fals
10. Care dintre următoarele NU reprezintă un client X?
    - ❑ Kcalc
    - ❑ gnome-terminal
    - ❑ Xorg
    - ❑ kdm

Capitolul 14

Utilitare pentru dezvoltare

First learn computer science and all the theory. Next develop a programming style. Then forget all that and just hack.

George Carrette

Ce se învață din acest capitol?

- Convenții de codare
- Editorul Vim
- Sisteme de control a versiunii; Subversion și Git
- Parcurgerea rapidă a codului
- Compilarea unei aplicații din surse
- Zonele unui executabil și proces
- Depanarea unui program folosind GDB, ddd și Valgrind
- IDE-uri: Eclipse, Anjuta, Microsoft Visual Studio
- Gestiunea unui proiect software

14.1 Introducere

Dezvoltarea aplicațiilor este una din principalele activități ale specialiștilor în calculatoare. Procesul de dezvoltare necesită un set de utilitare pe care programatorul (dezvoltatorul software le folosește), de la editoare și utilitare de parcurgere a codului, până la utilitare de depanare și investigare a execuției aplicației. Fără a își propune să acopere la nivel de detaliu aceste utilitare, capitolul de față prezintă principalele utilitare folosite de un dezvoltator de software pe un sistem Unix/Linux. Multe dintre acestea sunt portate pe Windows și Mac OS X. În general, dezvoltarea de aplicații pe un sistem Linux presupune folosirea de utilitare dedicate (editoare, compilatoare, depanatoare, mecanisme de documentare), în vreme ce dezvoltatorii de aplicații pe sisteme Windows folosesc, de obicei, un mediu integrat de dezvoltare (IDE – *Integrated Development*

Environment); cel mai cunoscut este Microsoft Visual Studio (vezi secțiunea 14.12.1). Totuși, și în Linux sunt folosite IDE-uri precum Eclipse sau Anjuta (vezi secțiunea 14.9).

14.2 Coding style (indent, astyle)

Una dintre greșelile frecvente realizate de programatorii începători este credința că vor putea înțelege codul sursă scris de ei înșiși oricând în viitor[1]. Scrierea unui cod dezorganizat, fără comentarii este, de obicei, echivalentă cu scrierea unui cod nementenabil.

Pentru a preveni situațiile în care se consumă timp pentru înțelegerea codului scris de altcineva trebuie respectate un set de norme de bază pentru editarea codului. Folosirea acestor norme ajută și lucrul într-o echipă de dezvoltare, când mai mulți dezvoltatori ajung să parcurgă codul scris de un altul.

Diverse proiecte software colectează normele preferate (sau obligatorii) de redactare a codului într-un document de stil de codare (*coding style*, *programming style*, *coding standards*, *code conventions*). Exemple sunt Java[2], GNU[3], BSD[4], Linux[5], Python[6], PHP[7] etc.

Stilurile de codare variază în funcție de limbajul de programare sau de proiect, dar există un set de reguli comune care facilitează înțelegerea și îmbunătățirea ulterioară a codului. Exemple de astfel de reguli sunt:

- folosirea de nume relevante pentru variabile; se preferă denumirea `num_files` în loc de `num_var`, `EncryptMessage` în loc de `DoStuff` sau `tmp` în loc de `TemporaryVariabileForStoringIntegerArraySize`; variabilele de genul `i`, `j` se consideră implicite pentru parcurgerea vectorilor iar folosirea lor nu trebuie detaliată;
- indentarea codului; corpul unui bloc de instrucțiuni se indentează fie cu TAB fie cu spații;
- spații/linii libere; pentru lizibilitate se recomandă folosirea spațiilor libere între argumentele unei funcții, înainte și după simbolul = în momentul inițializării unei variabile etc. și a liniilor libere pentru separarea blocurilor de instrucțiuni;
- funcții kilometrice; corpul funcțiilor trebuie limitat pentru a fi ușor de parcurs; funcții care depășesc două ecrane trebuie evitate;
- comentarii; codul trebuie comentat pentru a facilita înțelegerea acestuia; comentariile nu trebuie să fie redundante, trebuie să ajute programatorul acolo unde codul este dificil de înțeles;

[1] Eagleson's Law of Programming: Any code of your own that you haven't looked at for six or more months, might as well have been written by someone else.

[2] http://java.sun.com/docs/codeconv/

[3] http://www.gnu.org/prep/standards/

[4] http://www.freebsd.org/cgi/man.cgi?query=style&sektion=9

[5] http://lxr.linux.no/#linux+v2.6.31/Documentation/CodingStyle

[6] http://www.python.org/dev/peps/pep-0008/

[7] http://pear.php.net/manual/en/standards.php

- consecvența stilului; este mai bine să se scrie un cod greu lizibil, dar care are o consecvență în stil – chiar ilizibil – decât să se folosească mai multe stiluri, fie ele și bune.

indent, astyle În situațiile în care fișierele sursă conțin cod indentat necorespunzător, se pot folosi utilitare precum `indent` sau `astyle`.

Utilitarul `indent` indentează corespunzător codul C, conform unor opțiuni configurabile prin argumente: spațierea și indentarea folosită, introducerea de linii libere, afișarea codului de funcții, pentru blocuri `if` și `for`, afișarea comentariilor.

`indent` permite integrarea diverselor opțiuni în ceea ce se numește stil (`COMMONSTYLES`). În mod implicit se folosește stilul de codare folosit de GNU[1]. Fișierul 14.1 conține cod indentat necorespunzător. Folosind comanda de mai jos se va obține
fișierul 14.2 formatat folosind stilul GNU.

```
1 razvan@valhalla:~/development$ indent ugly-code.c -o ugly-code-default.c
```

Opțiunea `-o` este folosită pentru a specifica fișierul de ieșire. În absența acesteia se modifică direct fișierul sursă.

```
1  #include<stdio.h>
2
3  int main(){ int i;
4   int par=0,impar=0;
5   int   sum= 0;
6   for(i=0;i<100;i++){
7  sum=sum+i;
8  printf("partial␣sum␣is␣%d\n",sum);
9     if(sum%2==0)
10      { printf("par\n");
11        par++;}
12     else {printf("impar\n"); par--;}
13   }
14   return 0;}
```

Listing 14.1: Fișier indentat necorespunzător

```
1  #include<stdio.h>
2
3  int
4  main ()
5  {
6    int i;
7    int par = 0, impar = 0;
8    int sum = 0;
9    for (i = 0; i < 100; i++)
10     {
11       sum = sum + i;
12       printf ("partial␣sum␣is␣%d\n", sum);
13       if (sum % 2 == 0)
14         {
15           printf ("par\n");
16           par++;
```

[1] http://www.gnu.org/prep/standards/

```
17          }
18        else
19          {
20            printf ("impar\n");
21            par--;
22          }
23      }
24    return 0;
25  }
```

Listing 14.2: Fișier formatat cu stilul GNU

Folosind alte opțiuni se pot specifica alte stiluri de coding: `-kr` pentru stilul Kernighan & Ritchie, `-linux` pentru stilul Linux, `-orig` pentru stilul inițial BSD.

Fișierul 15.8 este formatat folosind stilul Linux, folosind comanda:

```
razvan@valhalla:~/development$ indent -linux ugly-code.c -o ugly-code-linux.c
```

```
#include<stdio.h>

int main()
{
        int i;
        int par = 0, impar = 0;
        int sum = 0;
        for (i = 0; i < 100; i++) {
                sum = sum + i;
                printf("partial sum is %d\n", sum);
                if (sum % 2 == 0) {
                        printf("par\n");
                        par++;
                } else {
                        printf("impar\n");
                        par--;
                }
        }
        return 0;
}
```

Listing 14.3: Fișier formatat cu stilul Linux

Utilitarul `astyle`[1] (*Artistic Style* permite indentarea și formatarea codului C, C++, C# și Java. La fel ca și `indent`, `astyle` are diverse opțiuni de formatare și stiluri predefinite. O lipsă a utilitarului astyle este absența opțiunilor de introducere de spații. Comenzile de mai jos formatează un fișier conform stilurilor GNU, Linux sau Kernighan & Ritchie:

```
razvan@valhalla:~/development$ astyle --style=gnu < ugly-code.c > ugly-code-astyle-gnu.c

razvan@valhalla:~/development$ astyle --style=linux < ugly-code.c > ugly-code-astyle-linux.c

razvan@valhalla:~/development$ astyle --style=k\&r < ugly-code.c > ugly-code-astyle-kr.c
```

[1] http://astyle.sourceforge.net/

14.3 Editorul Vim

Editoarele reprezintă utilitarele de bază pentru un dezvoltator de aplicații. Editoarele profesioniste vor oferi acestuia facilități pentru parcurgerea, prelucrarea și editarea rapidă a fișierelor cod sursă. Editoarele profesioniste vor oferi funcționalități de colorare a sintaxei (*syntax highlighting*), colapsarea codului (*code folding*), indentare automată etc.

În lumea Unix, cele mai cunoscute editoare sunt Vim și GNU Emacs. Secțiunea următoare prezintă modul de lucru și funcționalitățile editorului Vim.

Unul dintre cele două editoare protagoniste, tradițional, ale "războiului editoarelor"[1] este **`vi`**. Pentru că programul original nu mai este activ dezvoltat, au apărut numeroase clone, derivate ale codului, sau rescrieri complete. Cel mai cunoscut astfel de program este **`Vim`**[2], dezvoltat de Bram Moolenaar.

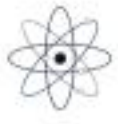
Majoritatea distribuțiilor de Linux vin cu o clonă de **`vi`** minimală preinstalată. Pentru majoritatea exemplelor următoare, aceasta nu este suficientă; recomandăm, deci, instalarea pachetului `vim` întreg. În sistemele Debian, de exemplu, pachetul instalat implicit se numește `vim-tiny`, iar cel cu mai multe opțiuni – `vim`. Pentru varianta care include interfață grafică, există `vim-full`.

Pornirea editorului. Moduri

Vim poate fi pornit din linie de comandă, având drept parametru opțional numele fișierului pe care să îl încarce. Lipsa acestui parametru înseamnă editarea unui fișier nou.

Odată pornit, editorul nu funcționează convențional. El este în modul **Normal**, unde nu este posibilă introducerea textului. Funcție de sarcina care trebuie îndeplinită, Vim poate fi într-unul dintre modurile de mai jos[3]:

- **Normal** – facilitează deplasarea rapidă prin text. De asemenea, o serie de opțiuni avansate ca mark-uri sau fold-uri, descrise sumar în continuare, pot fi manevrate în modul Normal.
- **Insert** – singurul mod în care putem introduce text. Acesta **nu** este modul implicit la pornirea editorului, deci nu este posibilă introducerea de text imediat după încărcarea unui fișier. Această particularitate este sursa multor confuzii. Pentru a intra în modul Insert din Normal, comanda cea mai simplă este `i`. Pentru a reveni în modul normal, se folosește `Escape`.

  Pentru a edita text eficient, nu se rămâne în modul Insert mai mult decât este nevoie.
- **Command** – comenzile avansate se introduc utilizând acest mod. Din modul Normal, se folosește `:` pentru a intra în modul Command. Lansarea comenzii se face folosind `Enter`, iar anularea ei, natural, prin `Escape`. Multe dintre comenzile Vim sunt precedate de `:`. Aceasta înseamnă că introducerea lor se face în modul Command.

[1] http://en.wikipedia.org/wiki/Editor_war
[2] Vi iMproved – http://www.vim.org/
[3] lista nu este exhaustivă

- **Visual** – selectarea unei porțiuni arbitrare de text se face folosind acest mod. După selectare, o comandă va acționa, dacă acest lucru este posibil, asupra textului selectat. Intrarea în modul Visual se face, din Normal, folosind `v`, iar revenirea în Normal – utilizând `Escape` sau lansând o comandă de editare.

Câteva comenzi de bază sunt:

- `:w` – salvare fișier ("write"). Dacă fișierului îi este asociat un nume, se suprascrie vechiul fișier. Dacă comanda este urmată de un nume de fișier, acesta este creat sau, dacă există, suprascris;
- `:q` – ieșire din program ("quit"). Dacă există modificări nesalvate, utilizatorul trebuie fie să le salveze (`:w`), fie să folosească varianta `:q!` pentru a nu le salva;
- `:wq` – scurtătură pentru cele două comenzi de mai sus.

Sistemul integrat de Help

Spre deosebire de majoritatea comenzilor Unix, pagina de manual `vim` nu își propune să documenteze în întregime utilizarea editorului, ci doar detalii despre argumentele în linia de comandă. Vim are un sistem intern de help, foarte dezvoltat.

Pentru a afla informații despre comanda `:w`, se folosește `:help :w`. Implicit, ecranul va fi împărțit în jumătate, cu partea superioară afișând informații de ajutor. Închiderea noii ferestre se face folosind `:q`.

Este util de reținut că se poate folosi `:help` pentru comenzi în mod Normal, comezi "extinse" (caracterul `:` care le precede trebuie inclus), dar și parametri de configurare. Aceștia din urmă trebuie să fie dați între ghilimele, de exemplu: `:help 'cindent'`. Nu în ultimul rând, o resursă excelentă pentru familiarizarea cu Vim este `vimtutor`, un tutorial *self-contained* care vine cu editorul. Lansarea lui se face convențional, în linie de comandă.

Deplasarea în modul Normal

Modul Normal oferă câteva opțiuni inedite de deplasare. Parte din puterea editorului `vi` stă în natura sa modală, pentru că atunci când nu se editează text, tastele alfabetice sunt disponibile pentru alte comenzi utile.

Deși este posibilă, inclusiv în modul Insert, folosirea tastelor săgeți pentru deplasare prin text, modul Normal oferă o alternativă avantajoasă. Se pot folosi tastele `hjkl` în loc de săgeți, după topologia:

```
1    k
2  h j l
```

Aceste taste au avantajul că, pe o tastatură standard QWERTY, se află pe *home row*, rândul central al tastaturii. Ele pot fi apăsate cu mâna dreaptă, fără a deplasa prea mult degetele din poziția lor naturală, folosind tehnica *touch typing*[1].

Alte comenzi utile pentru deplasare, precum și semnificațiile lor, sunt:

[1] http://en.wikipedia.org/wiki/Touch_typing

- `$` – sfârșitul liniei (analog `End`);
- `^` – începutul liniei (analog `Home`);
- `w` – cuvântul următor ("word");
- `b` – cuvântul anterior ("back");
- `fx` – următoarea apariție a caracterului `x` pe linia curentă ("find");
- `Fx` – precedenta apariție a caracterului `x` pe linia curentă;
- `<număr>g` – poziționare pe linia dată ("go to line");
- `gg` – prima linie;
- `G` – ultima linie;
- `%` – paranteza care corespunde celei de sub cursor. Perechile de paranteze pot fi configurate: `:help 'matchpairs'`.
- `/text` – următoarea apariție a textului specificat. Aceasta este o comandă "extinsă" (tastarea `/` provoacă intrarea în modul Command).

Editare avansată

Există comenzi care, lansate în modul Normal sau Visual, manipulează text. Majoritatea celor prezentate mai jos funcționează în felul următor:

- dacă o comandă este lansată în modul Normal, ea așteaptă o secvență de deplasare, pentru a specifica textul asupra căruia să opereze. Astfel, `dfa` va șterge tot textul din poziția curentă până la următoarea apariție a caracterului "a" pe linia curentă;
- dacă o comandă este lansată în modul Visual, ea acționează imediat asupra textului selectat. În plus, se iese din modul Visual și se revine în Normal. Excepție fac comenzile similare cu `c`, descrise mai jos, care duc în modul Insert.

Câteva comenzi de editare și variante utile ale lor sunt prezentate în continuare, cu mențiunea esențială că lista nu este nicidecum completă. Important este că o comandă poate fi urmată de orice mișcare[1], inclusiv folosind repetiții[2].

- `d` – ștergere ("delete"). Textul este pus în "clipboard";
  - `dd` – șterge linia curentă (scurtătură);
  - `D` – șterge restul liniei curente (scurtătură);
  - `dw` – șterge următorul cuvânt (comandă compusă);
  - `dG` – șterge de la linia curentă, până la finalul fișierului (comandă compusă);
- `c` – schimbare ("change"). Textul este pus în "clipboard" și editorul intră în modul Insert;
  - `cc` – schimbă linia curentă (scurtătură);

[1] `:h motion.txt`
[2] `:h repeat.txt`

  - `c` – schimbă restul liniei curente (scurtătură);
  - `c8w` – schimbă următoarele 8 cuvinte (comandă compusă);
  - `c^` – schimbă de la începutul liniei, până la caracterul curent (comandă compusă);
- `y` – copiere în "clipboard" ("yank"). Nu afectează textul, dar poate fi folosit în conjuncție cu `p`;
- `p` – inserare din "clipboard" ("put" sau "paste"). Pune conținutul ultimei comenzi `d`, `c`, sau `y` după poziția curentă.

O greșeală frecventă este aceea de a copia ceva folosind `y`, apoi, înainte de a `p`une conținutul în alt loc, a șterge conținut superfluu, spre exemplu o linie goală. Această ștergere suprascrie conținutul "clipboard"-ului. O soluție imediată este comandă `u` ("undo"), dar răspunsul mai amplu este folosirea registrelor.

Registre

Registrele sunt locuri în memorie unde Vim poate stoca text. Astfel, ceea ce până acum am numit "clipboard" este, de fapt, un registru implicit.

Registrele de uz general sunt adresate folosind literele alfabetului. Nu există nicio diferență între a folosi litere mici sau mari. Modul de folosire a registrelor cu comenzile de editare este următorul:

```
1 "Rc
```

Semnificația componentelor comenzii este următoarea:

- `"` – anunță că vom folosi un registru nestandard;
- `R` – anunță registrul folosit. Poate fi orice registru din domeniul `a-z` sau unul special[1];
- `c` – comanda propriu-zisă. Toate precizările și exemplele secțiunii precedente sunt valabile.

Utilizare `make`

Folosind comanda `:make`, se poate rula **`make`** (vezi secțiunea 11.5) direct din Vim. Implicit, Vim recunoaște mesajele de eroare și le listează, facilitând deplasarea prin surse.

Deplasarea între mesajele de eroare se face utilizând comanzile `:cnext` și `:cprev`. Aceste comenzi funcționează și dacă eroarea nu se află în fișierul curent, deschizând fișiere noi dacă este nevoie. Astfel, Vim poate fi folosit eficient pe post de IDE (vezi secțiunea 14.9).

O altă comandă utilă în acest sens este `:copen`, care deschide o fereastră cu toate erorile. Acesta poate fi folosită pentru poziționarea pe o anumită eroare fără a folosi `:cnext` în mod repetat. Denumirea generică a acestor facilități este `quickfix`.

[1] mai multe informații la `:h registers`

Configurarea Vim

Editorul Vim este remarcat prin flexibilitatea sa, iar elementul central al configurabilității sale sunt opțiunile. Folosind comanda `:set` se pot interoga sau configura opțiuni.

Unele opțiuni sunt de tip boolean. Acestea se setează pe *true* folosind numele lor (`:set autowrite`), iar pe false lipind `no` la începutul numelui (`:set noautowrite`).

O listă completă a lor este prezentă în help, dar poate fi intimidantă. Câteva exemple relevante sunt prezentate în continuare.

- `textwidth` – lungimea unei linii după care se face wrap. În general, pentru text, cod sursă, sau mesaje email, această opțiune trebuie să fie mai mică decât 80. O valoare de 0 înseamnă că Vim nu va face line wrap;
- `cindent` – dacă această opțiune este setată, Vim va face indentare "inteligentă" bazată pe sintaxa limbajului C. Aceasta este o opțiune booloeană;
- `filetype` – tipul fișierului editat. Această opțiune influențează tipul de *syntax highlighting* folosit, dar și alte opțiuni.

Nu în utimul rând, interogarea valorii unei opțiuni se face utilizând comanda `:set`, dar adăugând caracterul `?` după numele opțiunii:

```
1 :set cindent?
```

Configurare persistentă

Configurarea opțiunilor la fiecare rulare a editorului este incomodă. Pentru a păstra unele valori ale opțiunilor, dar și pentru a rula script-uri de extensie, se folosește un fișier de configurare.

Vim citește, în ordine, `/etc/vim/vimrc`, apoi `~/.vimrc`. Astfel, este posibil să suprascriem opțiuni globale folosind fișierul personal de configurare.

Distribuțiile livrează, în general, un fișier de configurare în `/usr/share/vim`. Acesta poate fi copiat în /etc/vim/vimrc și apoi personalizat local. Este important ca unele opțiuni, spre exemplu `syntax`[1], să fie activate tot timpul, pentru folosirea eficientă a editorului.

Există posibilitatea configurării unor opțiuni numai pentru anumite tipuri de fișiere. Linia următoare configurează lungimea liniei, indentarea, și comportamentul înlocuirii spațiilor cu tab-uri numai pentru sursele sau headerele C:

```
1 autocmd FileType c,cpp  set tw=72 cindent noexpandtab
```

Editarea de fișiere multiple

Vim poate edita mai multe fișiere simultan, iar abstracțiile pe care le folosește pentru a prezenta utilizatorilor acest lucru sunt diverse.

[1] `:syntax on`

O posibilitate este împărțirea ecranului în mai multe ferestre, cum face comanda `:help`. Deschiderea unei noi ferestre se face folosind `:split` sau `:vsplit`, urmate de numele fișierului de editat. Deplasarea între ferestre se face utilizând `hjk`, dar precedate de prefixul `Ctrl-W`. Închiderea unei ferestre este un simplu și bine-cunoscut `:q`.

Când mai multe ferestre nu sunt suficiente, vim poate folosi tab-uri. **Fiecare tab poate conține mai multe ferestre**, deci este o greșeală a asocia unu-la-unu fișierele deschise cu tab-urile. Pentru deschiderea unui tab se folosește `:tabnew`, urmat opțional de numele unui fișier, iar pentru deplasare între tab-uri `gt` și `gT`. Închiderea ultimei ferestre dintr-un tab provoacă închiderea tab-ului.

Informații mai complete se găsesc, ca de obicei, în help: `:help windows.txt` și `:help tabpage.txt`.

Lucrul cu proiecte mari

A edita un ansamblu de fișiere mari pune probleme complet diferite față de a adăuga câteva linii unui fișier de configurare. Vim are facilități pentru primul caz, așa cum se va vedea în continuare.

Pentru deplasarea într-un proiect C, Vim are suport de `ctags`[1]. Generarea unui fișier `ctags` se face folosind:

```
1 vlad@cormyr:~/school/so/tema4 $ ctags -aR *
```

Odată generat un astfel de fișier, Vim trebuie notificat de existența lui, prin intermediul opțiunii `tags`. Deplasarea la locul de definiție al funcției de sub cursor se face utilizând `Ctrl-]`, iar pentru revenirea în locul anterior[2] se folosește `Ctrl-T`. O alternativă mai puternică, dar ceva mai dificil de folosit, este `cscope`[3]. Există script-uri Vim care oferă facilități similare pentru cod scris în alte limbaje. În general, ele sunt publicate pe site-ul central Vim[4].

Uneori, este utilă memorarea unui loc într-un fișier. Vim numește acest concept **mark**. Pentru a pune un mark pe linia curentă, se folosește, în modul Normal, `m`, urmat de litera corespunzătoare mark-ului. Literele mici sunt mark-uri per-fișier, iar cele mari sunt globale. Deplasarea până la un mark se face utilizând ' (apostrof) urmat de litera corespunzătoare.

Combinația '' (două apostroafe) este analogul "Alt-Tab". Ea duce cursorul unde a fost înainte, și are euristici care detectează deplasările semnificative. Astfel, deplasările mici sunt ignorate.

O altă facilitate pentru lucrul cu fișiere lungi este **folding**. Aceasta este, în esență, comprimarea *vizuală* (fișierul rămâne neschimbat) a unei regiuni de text. Criteriul după care se face comprimarea poate fi nivelul de indentare, elemente de sintaxă, sau marcaje *ad hoc*. Modul de comprimare este dat de opțiunea `foldmethod`.

Combinații de taste esențiale pentru *folding* sunt:

[1] http://ctags.sourceforge.net/
[2] *popping the tag stack*
[3] http://cscope.sourceforge.net
[4] http://www.vim.org/scripts/index.php

- `zo` – deschide un fold;
- `zc` – închide un fold;
- `zf` – creează un fold, dacă `foldmethod=marker`. Această comandă inserează marcajele configurate cu `foldmarker`.

14.4 Sisteme de control al versiunii

14.4.1 Principii

Sistemele de control al versiunii sunt programe care țin codul sursă și toată istoria modificărilor asupra lui. Folosirea unui astfel de sistem este o practică aproape universală în industria software, dar și o deprindere utilă pentru proiecte personale.

Un sistem de control al versiunii (VCS – *Version Control System*) protejează, în primul rând, împotriva modificărilor care au efecte neprevăzute, dar sunt greu de observat. Dacă, în urma unui build, se constată că s-a introdus un bug, se pot inspecta schimbările făcute și determina mai simplu sursa erorii.

O altă situație care poate apărea este căderea hardware sau ștergerea din greșeală a unor fișiere. Un VCS oferă un backup permanent sincronizat al codului, deci pierderile ar putea fi minimizate.

Nu în ultimul rând, un VCS facilitează colaborarea între dezvoltatori, prin combinarea automată a schimbărilor asupra aceluiași fișier, acolo unde acest lucru este posibil.

Câteva elemente cheie în jargonul VCS sunt:

- **repository** – locul central unde sunt ținute sursele și istoria. În majoritatea cazurilor, repository-ul se află pe un server și este foarte bine protejat. Pierderea unui repository poate fi o catastrofă. Sistemele distribuite (vezi secțiunea 14.4.3) redefinesc noțiunea de repository;
- **working copy** – copia unui repository deținută de un dezvoltator. De obicei, un working copy nu are istoria codului, deci pentru consultarea lor este nevoie de comunicația cu repository-ul. Obținerea unui working copy se face printr-o operație numită tradițional **checkout**;
- **commit** – un set de modificări pe care un dezvoltator îl trimite către repository. Istoria codului este o înșiruire de commit-uri;
- **update** – sincronizarea unui working copy cu repository-ul. Această operație va aplica commit-urile care nu sunt deja în working copy;
- **branch** – o istorie distinctă a sursei. Conceptul este folosit tradițional pentru experimente asupra codului sursă, dar sistemele distribuite redefinesc acest concept.
- **tag** – o stare semnificativă a repository-ului. Un exemplu de tag este un release. O altă situație în care ar fi nevoie de un tag este înaintea unei refactorizări importante.

Se disting două tipuri de sisteme de versionare:

- Sisteme centralizate. Aceastea sisteme se pretează perfect la descrierile conceptelor prezentate mai sus. Sunt încă foarte folosite și activ dezvoltate, dar tendința actuală este migrarea către sistemele distribuite. Exemple relevante sunt CVS, Subversion, Perforce.
- Sisteme distribuite. Ideea din spatele lor este aceea că fiecare working copy are, de fapt, istoria completă a codului. Astfel, nu mai există un repository central. Asemenea sisteme sunt descentralizate, în sensul că nu sunt coerente toate copiile dezvoltatorilor. Din această cauză, o copie a codului se numește branch. Pentru că nu există un repository central, branch-urile sunt adesea schimbate prin email. Exemple relevante de VCS distribuite sunt: git, darcs, Mercurial, Bazaar.

În VCS se țin toate fișierele care nu pot fi generate altfel. Cu alte cuvinte, sistemele VCS mențin surse, dar nu și fișiere binare. O altă regulă este ca, în orice moment, starea repository-ului să fie consistentă; codul trebuie tot timpul să compileze, indiferent dacă are bug-uri sau facilități incomplete. Sintagma "a strica build-ul" descrie un commit care lasă codul într-o stare în care nu se compilează.

14.4.2 Subversion

Subversion[1] este sistemul centralizat cel mai folosit în momentul de față. Este urmașul CVS și adoptă o abordare clasică, simplu de înțeles. Executabilul de Subversion se numește **svn**.

Operația prin care se crează un working copy este `svn checkout`. Subversion poate funcționa peste SSH, sau poate folosi protocolul propriu.

```
1 vlad@cormyr $ svn checkout svn://svn.rosedu.org/wouso
```

Pentru a face un nou commit, se folosește `svn commit`. Schimbările sunt automat transmise serverului central, deci computerul are nevoie de conexiune la rețea. Înainte de a face un commit, se recomandă actualizarea working copy-ului pentru a evita conflictele. Aceast lucru se face folosind comanda `svn update`.

Adăugarea unui nou fișier la repository se face folosind `svn add`. Transferul către server nu se va face, însă, decât la următorul commit.

Folosind `svn status`, se poate inspecta starea copiei de lucru. Această comandă tipărește un sumar al fișierelor schimbate sau adăugate, dar și al celor care nu sunt în repository-ul central ("untracked"). Pentru a vedea schimbările aduse fișierelor, se folosește `svn diff`.

Subversion nu are suport nativ pentru branch-uri și tag-uri, fapt pentru care majoritatea repository-urilor conțin, în rădăcină, 3 directoare:

- `branches` – simulează funcționarea branch-urilor din alte VCS-uri. Practic, crearea unui branch este o copiere a surselor într-un nou subdirector al `branches`;
- `tags` – simulează existența tag-urilor. Crearea unui tag este echivalentă cu o copiere, cu mențiunea că există o convenție ca nimeni să nu modifice conținutul directorului `tags`

[1] http://subversion.tigris.org/

- `trunk` – aici rezidă codul propriu-zis. Acest director este sursa copierilor care fac tag-uri și branch-uri.

14.4.3 Git

Git[1] este un VCS distribuit dezvoltat, inițial, de Linus Torvalds. Se distinge prin flexibilitate și viteză și are o comunitate activă. Este sistemul de versionare folosit de kernelul Linux.

Pentru a folosi eficient Git, se recomandă activarea culorilor și configurarea globală a numelui și adresei de mail a utilizatorului.

```
1 vlad@cormyr $ git config --global user.name "John Doe"
2
3 vlad@cormyr $ git config --global user.email "john.doe@foobar.com"
4
5 vlad@cormyr $ git config --global color.ui always
```

Operația de creare a unui "working copy" (numit branch în terminologie Git) este `git clone`. Pentru comunicația cu serverul se pot folosi HTTP sau SSH, dar este recomandat protocolul propriu Git, care are suport mai bun de compresie.

```
1 vlad@cormyr $ git clone git://git.rosedu.org/rtt.git
```

După modificarea fișierelor, ele trebuiesc adăugate explicit și apoi creat un commit. În cultura Git, acest lucru se sumarizează prin "Git tracks changes, not files". `git commit` lansează un editor în care este solicitată o descriere a noului commit.

```
1 vlad@cormyr $ git add binops.c
2
3 vlad@cormyr $ git commit
```

Starea branch-ului curent poate fi inspectată folosind `git status`, iar schimbările pot fi propagate către branch-ul de unde s-a făcut clonarea ("origin") folosind `git push`. Sincronizarea cu origin se face folosind `git pull`.

Aspecte netriviale ale utilizării Git sunt lucrul cu branch-uri, operațiile de merge, rezolvarea conflictelor, sau folosirea `git rebase`.

14.5 Analiza și parcurgerea codului

Un dezvoltator software profesionist va petrece, de regulă, mult mai mult timp în parcurgerea, înțelegerea și revizia fișierelor cod sursă decât în crearea de cod. Altfel spus, operația de citire a codului este mai frecventă și mai de durată decât cea de scriere.

Pentru a facilita înțelegerea codului, cel care scrie codul va trebui să urmărească un stil de codare sau un set de norme. În același timp, însă, cel care urmărește sau revizuiește codul dispune de un set de utilitare specifice acestui scop. În momentul în care proiectul software dispune de multe fișiere sursă, este foarte important ca revizorul să ajungă rapid la anumite poziții din cadrul fișierelor sursă.

[1] http://git-scm.com/

Utilitare precum `ctags` ajută revizorul să ajungă în locul unde a fost definită o funcție sau o variabilă din doar câteva combinații de taste. Revizorul poate urmări astfel setul de apeluri din cadrul unei aplicații, sau locurile în care o anumită variabilă este folosită sau utilizată. Unele IDE-uri (precum Microsoft Visual Studio (vezi secțiunea 14.12.1).

14.5.1 ctags

Utilitarul `ctags` este folosit pentru a genera un index al simbolurilor dintr-un set de fișiere sursă pentru parcurgerea ușoară a acestora. Indexul este generat într-un fișier **tag** care va fi ulterior folosit de un editor.

Utilitarul are asociate două comenzi: **`ctags`** și **`etags`**. Comanda **`ctags`** este folosită pentru a genera fișierul index pentru a fi folosit de editoare din familia **vi**, iar comanda **`etags`** generează fișier index pentru editoare din familia **Emacs**. Editoarele folosesc comenzi specializate pentru a găsi rapid locul în care este definită/utilizată o funcție sau variabilă pe baza fișierului index.

În general, pentru a genera un fișier index se folosește opțiunea `-R` pentru a parcurge recursiv toate fișierele sursă din directorul curent. Astfel, pentru a genera fișierele index pentru sursele bibliotecii standard C, se vor folosi comenzile:

```
1 razvan@valhalla:~/packages/glibc-2.7/glibc-2.7$ ctags -R .
2
3 razvan@valhalla:~/packages/glibc-2.7/glibc-2.7$ etags -R .
4
5 razvan@valhalla:~/packages/glibc-2.7/glibc-2.7$ ls -l tags TAGS
6 -rw-r--r-- 1 razvan razvan 5338062 Sep 26 11:31 TAGS
7 -rw-r--r-- 1 razvan razvan 7364195 Sep 26 11:31 tags
```

În mod implicit, **`ctags`** generează fișierul `tags`, iar **`etags`** generează fișierul `TAGS`. Fișierul este încărcat automat de editor în cazul în care fișierul index se găsește în directorul curent. Altfel, fișierul poate fi încărcat în Vim folosind o comandă de forma

```
1 :set tags+=/usr/include/tags
```

sau în Emacs, folosind

```
1 M-x visit-tags-table
```

Detalii despre utilizarea fișierelor index în Vim pentru parcurgerea rapidă a codului se găsesc în secțiunea 14.5.1. Pentru Emacs, informații utile sunt descrise în secțiunea asociată a manualului[1].

14.5.2 Analiza statică a codului – splint

Pe lângă erorile raportate de compilator, sau erorile/problemele din momentul execuției raportate de utilitare precum Valgrind (vezi secțiunea 14.8.2), dezvoltatorul poate folosi utilitare de analiză statică a codului (*Static code analysis*[2]). Aceste utilitare parcurg codul sursă fără compilarea, interpretarea sau executarea acestuia.

[1] http://www.gnu.org/software/emacs/manual/html_node/emacs/Tags.html#Tags
[2] http://en.wikipedia.org/wiki/Static_code_analysis

Un astfel de utilitar este **Splint**[1]. Descendent al utilitarului **lint** din Unix, Splint permite detectarea potențialelor probleme de securitate și a erorilor de codare. Splint poate fi configurat să folosească diverse niveluri de verificare. Conform paginii de manual, se oferă un premiu special persoanei care creează un program util care nu raportează erori în cazul folosirii opțiunii `-strict` (verificare strictă).

Un exemplu de rulare a comenzii **`splint`** și o parte a ieșirii acesteia este prezentat mai jos:

```
1  razvan@valhalla:/tmp$ splint -warnposix mini-shell.c
2  Splint 3.1.2 --- 20 Feb 2009
3
4  mini-shell.c: (in function simple_cmd)
5  mini-shell.c:38:5: Return value (type int) ignored: close(pin[0])
6    Result returned by function call is not used. If this is intended, can
   cast
7    result to (void) to eliminate message. (Use -retvalint to inhibit
   warning)
8  mini-shell.c:52:7: Test expression for if not boolean, type int:
9                        cmd->io_flags \& 0x01
10   Test expression type is not boolean or int. (Use -predboolint to
   inhibit
11   warning)
12 mini-shell.c:65:5: Assignment of int to char: str[0] = 0
13   Types are incompatible. (Use -type to inhibit warning)
14 mini-shell.c:68:48: Possibly null storage passed as non-null param:
15                       strlen (getenv(par->string))
16 mini-shell.c:25:5: Function exported but not used outside mini-shell:
17                     simple_cmd
18   A declaration is exported, but not used outside this module.
   Declaration can
19   use static qualifier. (Use -exportlocal to inhibit warning)
20    mini-shell.c:171:1: Definition of simple_cmd
21 [...]
```

După cum reiese din exemplul de mai sus, câteva dintre probleme de codare, neraportate implicit de compilator, sunt:

- ignorarea valorii de retur a unei funcții;
- folosirea unei expresii care întoarce o valoare întreagă în locul unei valori booleene;
- inițializarea unei variabile cu o valoare de alt tip;
- posibila folosire a unei valori null;
- definirea unei funcții non-statice care nu este folosită într-un alt modul.

O bună parte din problemele raportate de Splint nu sunt relevante în contextul unui program. De aceea, Splint pune la dispoziția dezvoltatorului opțiuni de dezactivare a anumitor mesaje, precum `-exportlocal`, `-predboolinit`, `-retvalint` etc.

În general, informațiile oferite de Splint pot fi foarte stricte și fără importanță deosebită în cadrul programului. De aceea, se recomandă selectarea informațiilor furnizate după relevanța acestora.

[1] http://www.splint.org/

14.6 Automatizarea compilării

Este posibil ca unele aplicații să nu se găsească în repository-urile distribuției folosite. Mai mult, este posibil ca acestea să nu fie disponibile într-un format ce ar facilita instalarea (cum ar fi .deb, .rpm). Astfel de aplicații trebuie compilate explicit pentru platforma folosită, pornind de la sursele lor.

14.6.1 Compilarea unei aplicații din surse

De cele mai multe ori, sursele unei aplicații se găsesc în formatul tar.gz sau tar.bz2. Așadar, primul pas este extragerea surselor din aceste arhive folosind utilitarul tar. Presupunând că arhiva cu sursele se numește foo.tar.gz sau foo.tar.bz2, pentru a extrage sursele trebuie rulată una din comenzile:

```
1 user@sys:~$ tar -xzvf foo.tar.gz
2 [...]
3 user@sys:~$ tar -xjvf foo.tar.bz2
```

Următorul pas îl reprezintă obținerea fișierului makefile. În acest punct este recomandată citirea fișierelor README sau INSTALL pentru a vedea exact pașii ce trebuie urmați. Deși pot apărea mici diferențe, majoritatea aplicațiilor pot fi instalate urmărind pașii descriși în continuare.

Scriptul configure este cel responsabil de generarea makefile-ului, acesta verificând versiunile de software disponibile și dacă diverse dependințe sunt îndeplinite. Este important de menționat că prin modificarea acestui script sau prin parametrii pasați în linia de comandă, se poate specifica directorul în care urmează să fie instalată aplicația. Scriptul se găsește în directorul în care am extras sursele, și rularea lui se face astfel:

```
1 user@sys:~/foo$ ./configure
```

Orice fel de eroare întâlnită la acest pas, cum ar fi lipsa unei alte aplicații, va fi semnalată în output-ul comenzii și trebuie rezolvată înainte de trecerea la următoarea etapă.

În urma scriptului de configurare a rezultat un fișier numit makefile (sau Makefile, depinde de fișierul de configurare), pe care îl vom folosi pentru a compila sursele. Pentru a face acest lucru vom folosi utilitarul make, care caută în directorul curent un fișier makefile pe care îl va folosi în obținerea executabilelor:

```
1 user@sys:~/foo$ make
```

Astfel, după rularea make, sursele vor fi compilate însă executabilele rezultate se află doar în directorul curent. Pentru a le muta în directorul de instalare:

```
1 user@sys:~/foo$ make install
```

În acest moment, aplicația a fost instalată și dacă directorul în care aceasta se află există în PATH, ar trebui să se poată rula din directorul curent.

Pentru a scăpa de fișierele temporare ce au fost generate în procesul de configurare și compilare, se poate rula:

```
1 user@sys:~/foo$ make clean
```

Pentru dezinstalarea aplicației, pot fi șterse manual fișierele din directorul unde a fost instalată, sau se poate folosi makefile-ul generat anterior, rulându-se:

```
1 user@sys:~/foo$ make uninstall
```

GNU Autotools

GNU Autotools[1] reprezintă un set de utilitare din cadrul proiectului GNU ce ajută la creșterea portabilității pachetelor software și a simplificării procesului de instalare a acestora. Cum am văzut anterior, majoritatea aplicațiilor pot fi instalate urmând cei trei pași simpli : ./configure, make, make install. Acest lucru se datorează în mare parte utilitarelor din Autotools, care reușesc să uniformizeze procesul de build, indiferent de platformă.

Utilitarele care fac parte din Autotools sunt:

- Automake – pornind de la un fișier numit Makefile.am, creează un fișier Makefile.in, folosit apoi de scriptul de configurare pentru a genera Makefile-ul final;
- Autoconf – este cel care creează scriptul de configurare, pornind de la un fișier numit configure.ac;
- Libtool – abstractizează procesul de creeare a librăriilor statice și dinamice.

14.7 Execuția unui program

În urma procesului de compilare și link-editare rezultă un executabil (vezi secțiunea 11.1.2). Acest executabil deține informații necesare sistemului de operare la rulare, pentru crearea unui proces.

14.7.1 Zonele de memorie ale unui executabil și proces

Un fișier executabil are un format specific. Pe lângă datele de organizare internă, un executabil conține informații despre instrucțiunile care vor fi executate pe procesor și datele folosite. Aceste informații se vor stoca în memorie în momentul creării și rulării procesului asociat. Un executabil va conține, așadar, **zona de cod** și **zonele de date**.

Zona de cod, denumită și zona text este, de fapt, traducerea în cod mașină a funcțiilor scrise în cod sursă. Aceste instrucțiuni vor fi executate pe rând pe procesor, în momentul creării procesului. Funcția main dintr-un program C este punctul de start din zona de cod.

Zonele de date reprezintă echivalentul binar al variabilelor globale definite de un programator în codul sursă. Există mai multe zone de date:

- `zona pentru date inițializate` (*.data*) unde sunt stocate variabilele globale inițializate (de forma `int size = 30;`)

[1] http://sourceware.org/autobook/autobook/autobook.html

- `zona pentru date neinițializate` (`.bss`) unde sunt stocate variabilele globale neinițializate (de forma `int file_array[10];`)
- `zona pentru date read-only` (`.rodata`) unde sunt stocate datele ce pot fi doar citite; din această categorie fac parte literalii din C; de exemplu, în cazul instrucțiunii `printf("Hello, World!\n");`, șirul `Hello, World\n` este stocat în `.rodata`.

În momentul creării procesului, se alocă memorie pentru zonele de mai sus. În afara acestor zone, în cadrul unui proces se alocă zone de memorie doar pe parcursul rulării (zone de memorie dinamice), care nu sunt disponibile în executabil. Cele mai importante zone sunt stiva și heap-ul.

Stiva (*stack*) este zona de memorie dinamică folosită pentru a reține informații despre apelurile de funcții și pentru a stoca variabilele locale funcțiilor. La fiecare apel de funcție, pe stivă se creează un cadru de stivă (*stack frame*) cu informații despre apel și pentru alocarea variabilelor locale funcției. În momentul în care se revine din funcții, cadrul de stivă asociat este eliberat de pe stivă. Motivul pentru care se recomandă evitarea apelurilor recursive este încărcarea stivei în cazul unui număr mare de apeluri.

Heap-ul este zona de memorie folosită pentru alocări dinamice. Alocările dinamice sunt alocările care se petrec în timpul rulării procesului (la *runtime*). Opusul este dat de alocările statice, care sunt realizate în momentul compilării. Variabilele globale sunt alocate static în zona `.data` sau în zona `.bss`.

Alocarea dinamică se realizează, în C, cu ajutorul apelurilor de bibliotecă `malloc`, `calloc` sau `realloc`. Acestea primesc ca argument numărul de octeți pentru regiunea care va fi alocată. O greșeală frecventă este omiterea eliberării zonelor de memorie alocate. Orice zonă de memorie alocată folosind apelurile din familia malloc va trebui dezalocată folosind apelul `free` atunci când nu mai este folosită. Dacă se omite operația de eliberare a memoriei se poate ajunge la situații de *memory leaking*[1] care afectează negativ funcționarea aplicației sau a sistemului.

14.7.2 Utilizarea bibliotecilor partajate

În momentul execuției, sistemul de operare efectuează o serie de operații pentru a pregăti aplicația pentru rulare. Procesele lucrează în zone de memorie independente unele de altele. Astfel, ceea ce se găsește la o anumită adresă pentru un proces nu se găsește și la alt proces.

Pe lângă încărcarea zonei de cod și a zonei de date din executabil, sistemul de operare trebuie să încarce și bibliotecile partajate. Bibliotecile statice devin parte din aplicații și nu au cum să fie tratate la execuție.

Pentru a vizualiza lista bibliotecilor partajate utilizate de un program, se folosește comanda **`ldd`**, precum în exemplul următor:

```
1 mircea@cougar:~/carte-uso/cap-10$ ldd /bin/bash
2         linux-gate.so.1 =>  (0xffffe000)
3         libreadline.so.5 => /lib/libreadline.so.5 (0xb7eb4000)
4         libhistory.so.5 => /lib/libhistory.so.5 (0xb7eac000)
```

[1] http://en.wikipedia.org/wiki/Memory_leak

```
5         libncurses.so.5 => /lib/libncurses.so.5 (0xb7e6a000)
6         libdl.so.2 => /lib/libdl.so.2 (0xb7e66000)
7         libc.so.6 => /lib/libc.so.6 (0xb7d3a000)
8         /lib/ld-linux.so.2 (0xb7f0d000)
```

Informațiile despre bibliotecile partajate aflate în sistem sunt colectate (manual sau automat) utilizând programul **`ldconfig`**. Această comandă:

- caută biblioteci partajate în locațiile standard și în directoarele specificate în /etc/ld.so.conf,
- configurează legături simbolice corecte în directoarele cu biblioteci (pentru a oferi denumiri standard biblitecilor, din punct de vedere al formatului)
- salvează referințele către biblioteci într-un cache.

Rularea programului **`ldconfig`** la fiecare bootare este ineficientă – acesta este un motiv în plus pentru utilizarea unui cache.

Modul în care programele utilizează bibliotecile de funcții este definit la link-editare. Linker-ul (**`ld`**) caută bibliotecile partajate în directoarele standard și în directoarele adăugate în linia de comandă prin intermediul unor parametri speciali (`-rpath dir` și `-Ldir`).

Într-un sistem bazat pe biblitecile standard GNU C, inclusiv toate sistemele Linux, pornirea unui executabil de tip ELF (formatul standard al executabilelor) determină încarcarea și rularea unui program loader. Pe sistemele Linux, acesta este `/lib/ld-linux.so.X` (unde X este versiunea). Acest loader găseste și încarcă bibliotecile partajate utilizate de către program.

Dacă se dorește supraîncărcarea unor funcții dintr-o bibliotecă dar cu păstrarea restului bibliotecii, se pot introduce bibliotecile supraîncarcate în `/etc/ld.so.preload` – aceste biblioteci "preîncărcate" vor avea întâietate relativ la setul standard. Acest fișier pentru preîncărcări este utilizat de obicei pentru patch-uri de urgență – distribuțiile nu includ de obicei un astfel de fișier.

14.7.3 Analiza apelurilor de sistem și a semnalelor

Pentru a interacționa cu sistemul de operare (cel care partajează resursele computerului), bibliotecile de funcții apelează o serie de funcții ale kernel-ului numite apeluri de sistem (system calls).

În continuare sunt prezentate doar câteva dintre utilizările apelurilor de sistem:

- deschiderea/inchiderea unui fișier (**`open`**, **`close`**)
- citirea/scrierea dintr-un/intr-un fișier (**`read`**, **`write`**)
- deschiderea unui socket (pentru o conexiune TCP/IP)
- executia unui program (**`execv`**)

Aceste funcții sunt implementate la nivelul nucleului sistemului de operare deoarece SO-ul face management-ul resurselor. Bibliotecile de functii "îmbracă" aceste apeluri în funcții/clase/obiecte utilizabile direct de către programatori.

strace

Comanda strace este utilă pentru urmărirea apelurilor de sistem efectuate la execuția unui program. Această analiză reprezintă o metodă destul de eficientă în debugging-ul programelor al căror cod sursa nu-l avem.

Pentru a observa ce informații poate să ofere **strace**, se va analiza ieșirea programului pentru o aplicație simplă (o parte din ieșirea programului a fost eliminată pentru a ușura analiza):

```
1  mircea@cougar:~/carte-uso/cap-10$ strace echo "USO"
2  execve("/bin/echo", ["echo", "USO"], [/* 51 vars */]) = 0
3  uname({sys="Linux", node="aquarium", ...}) = 0
4  [...]
5  access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or
   directory)
6  open("/etc/ld.so.cache", O_RDONLY)      = 3
7  fstat64(3, {st_mode=S_IFREG|0644, st_size=141679, ...}) = 0
8  [...]
9  close(3)                                = 0
10 open("/lib/tls/libc.so.6", O_RDONLY)    = 3
11 read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\220R\1"..., 512)
    = 512
12 fstat64(3, {st_mode=S_IFREG|0755, st_size=1363203, ...}) = 0
13 [...]
14 close(3)                                = 0
15 [...]
16 open("/usr/lib/locale/locale-archive", O_RDONLY|O_LARGEFILE) = 3
17 fstat64(3, {st_mode=S_IFREG|0644, st_size=38399616, ...}) = 0
18 [...]
19 close(3)                                = 0
20 [...]
21 fstat64(1, {st_mode=S_IFCHR|0600, st_rdev=makedev(136, 1), ...}) = 0
22 [...]
23 write(1, "USO\n", 4USO
24 )                       = 4
25 [...]
26 exit_group(0)                           = ?
```

Se poate observa:

- primul apel de sistem (**execve**) indică ce parametri a primit programul: echo și USO
- programul a încercat să deschidă fișierul `/etc/ld.so.preload` dar nu a fost găsit (scopul acestui fișier este descris în subcapitolul dedicat bibliotecilor)
- apelul de sistem **open** a returnat valoarea 3 – aceasta reprezintă un descriptor de fișier (fd – file descriptor) – descriptorul de fișier este folosit în procesele de citire și în procesele de scriere (e un fel de pointer la fișier, pointer utilizat de sistemul de operare)
- apelul de sistem **close** are ca singur parametru – descriptorul de fișier care trebuie închis – odată închis, un descriptor de fișier poate fi refolosit
- apelul de sistem **read** indică o citire din descriptorul de fișier 3
- apelul de sistem **write** are 3 parametri:

- 1 = file descriptor-ul unde se scrie: în cazul nostru, după cum știm, 1 este file descriptor-ul pentru stdout (ieșirea standard)
- USO\n = acesta este textul ce trebuie afișat de comanda echo
- 4 = lungimea textului ce trebuie afisat

În acest program s-au putut identifica câteva apeluri de sistem frecvente.

14.8 Depanarea unui program

De cele mai multe ori, programele scrise de noi nu funcționează corect de la prima rulare. De asemenea, se poate întâmpla ca un program care a funcționat corect timp de câțiva ani înainte să nu mai ruleze bine acum, în momentul introducerii unei date neprevăzute initial.

Depanarea unui program este o metodă de a găsi și identifica aceste erori sau defecte din program. Depanarea este o activitatea de durată și obositoare, de obicei. Abilitatea programatorului de a identifica *bug-urile* (greșelile din program) reprezintă factorul cheie în acest proces. Dar, nu trebuie uitată influența pe care o au metodele de depanare (*debugging*).

Desigur, pentru programele scurte se poate folosi *printf* pentru afișarea diverselor informații chiar dacă este o metodă ce presupune modificarea codului mai mult decât este nevoie pentru corectarea greșelii.

Pentru programele mai complicate este necesară folosirea unor instrumente de depanare, așa numitele *debuggers*. Acestea sunt programe sau biblioteci ce-i permit programatorului monitorizarea execuției unui program, oferindu-i capacitatea de a-l porni, opri, reporni și, în unele cazuri, a rula înapoi.

14.8.1 gdb, ddd

Un bun instrument de debug este GNU Debugger cunoscut mai mult sub denumirea prescurtată de GDB[1], depanatorul standard GNU, scris de Richard Stallman în 1986. Este un instrument portabil, capabil de a depana programe scrise într-o suită mai largă de limbaje (C, Ada, Basic, Fortran, C++ etc).

GDB se remarcă prin următoarele 4 lucruri, capacități cerute de la orice program de depanare a codului:

- se permite pornirea execuției unui program, specificând orice parametru ce-i poate afecta comportamentul;
- este posibilă oprirea programului într-un anumit punct, pe baza unor condiții specificate de utilizator;
- în cazul opririi premature a programului, se pot examina toate condițiile ce au dus la acest caz;

[1] http://sourceware.org/gdb

- se permite schimbarea valorii unei variabile – uneori putându-se schimba și instrucțiuni – pentru a modifica execuția programului fără a părăsi depanatorul.

Pentru a putea depana un program cu GDB, acesta trebuie întâi compilat utilizând un flag special *-g* pentru a se permite salvarea anumitor informații ce vor fi utile în continuare. De asemenea, se recomandă evitarea oricărei optimizări.

În continuare, vom prezenta un exemplu de rulare a unui program sub gdb. Pornim prin a lista problema pe care o avem:

```
mihai@keldon:~/tmp$ cat 1.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
        int i, n, **a, j;
        n = 6;
        a = (int*) malloc(n * sizeof(char));
        for (i = 0; i < n; i++){
                for (j = 0; j < n; j++)
                        a[i][j] = i + (j-1);
        }

        for (i = 0; i < n; i++){
                for (j = 0; j < n; j++)
                        printf("%d ", a[i][j]);
        printf("\n");
        }
        return 0;
}

mihai@keldon:~/tmp$ gcc -Wall 1.c
1.c: In function 'main':
1.c:8: warning: assignment from incompatible pointer type

mihai@keldon:~/tmp$ ./a.out
Segmentation fault
```

Deși este normal să tratăm întâi warningurile și apoi să depanăm codul, vom proceda diferit acum, ignorând outputul lui gcc.

```
mihai@keldon:~/tmp$ gcc -Wall -g -O0 1.c
...
mihai@keldon:~/tmp$ gdb ./a.out
...
(gdb) run
Starting program: /home/mihai/tmp/a.out

Program received signal SIGSEGV, Segmentation fault.
0x08048488 in main () at 1.c:11
11                              a[i][j] = i + (j-1);
(gdb) list
6               int i, n, **a, j;
7               n = 6;
8               a = (int*) malloc(n * sizeof(char));
(gdb) print *a
$4 = (int *) 0x0
(gdb) break main
```

```
18 Breakpoint 1 at 0x8048445: file 1.c, line 7.
19 (gdb) r
20 The program being debugged has been started already.
21 Start it from the beginning? (y or n) y
22 Starting program: /home/mihai/tmp/a.out
23
24 Breakpoint 1, main () at 1.c:7
25 7               n = 6;
26 (gdb) next
27 8               a = (int*) malloc(n * sizeof(char));
28 (gdb) display a
29 1: a = (int **) 0xbfe62388
30 (gdb) n
31 9               for (i = 0; i < n; i++){
32 1: a = (int **) 0x8634008
33 (gdb)
34 10                      for (j = 0; j < n; j++)
35 1: a = (int **) 0x8634008
36 ...
37 (gdb) q
38 The program is running.  Exit anyway? (y or n) y
```

În interiorul gdb, s-au folosit următoarele comenzi:

- run, prescurtat r – permite execuția programului;
- list, prescurtat l – permite listarea codului programului în jurul punctului unde acesta s-a oprit;
- print, prescurtat p – permite afișarea unei variabile;
- break, prescurtat b – permite stabilirea unui punct de întrerupere în interiorul programului;
- next, prescuratat n – execută următoarea instrucțiune a programului (cea afișată în momentul tastării comenzii);
- display, disp – permite afișarea unei variabile pe parcursul procesului de depanare;
- quit, q – terminarea depanării programului, ieșirea din gdb.

Observați că simpla trimitere a unei comenzi goale (doar apăsarea tastei Enter) duce la executarea comenzii anterioare.

Pentru mai mult detalii, consultați manualul GDB[1],.

ddd

Deși foarte util, GDB are un dezavantaj major: neavând interfață grafică, folosirea lui este mai greoaie. Din fericire, există un front-end grafic sub denumirea de Data Display Debugger (DDD[2]).

Pe lângă oferirea unei interfețe grafice ce permite stabilirea mult mai ușoară a unor puncte de oprire a programului (*breakpoints*), ddd s-a remarcat prin afișarea interactivă

[1] http://sourceware.org/gdb/current/onlinedocs/gdb_toc.html#SEC_Contents
[2] http://www.gnu.org/software/ddd/

a datelor programului utilizând grafuri. Această afișare permite studierea unui program prin analiza datelor și nu a liniilor de cod, un proces evident mult mai simplu.

```
mihai@keldon:~/tmp$ gcc -Wall -g -O0 1.c
...
mihai@keldon:~/tmp$ ddd ./a.out
...
```

Fiind un front-end al GDB, principiile de folosire sunt similare. Se va compila sursa și se va porni debugger-ul, restul comenzilor fiind înlocuite cu butoane în mediul grafic.

14.8.2 valgrind

Valgrind[1] este un utilitar de programare folosit pentru a depana memoria, a detecta leak-urile de memorie și a face profiling. Autorul lui este Julian Seward, care în 2006 a câștigat un al doilea Google-O'Reilly Open Source Award pentru Valgrind.

Valgrind este un framework de analiză dinamică. El conține un set de utilitare, iar fiecare dintre acestea realizează un tip aparte de depanare sau profiling. Dintre aceste componente, cele mai importante sunt:

- memcheck – checker de memorie.
- cachegrind – simulator de cache, numărul de instrucțiuni executate și cache miss-uri cauzate
- helgrind – depistează posibile race conditions din program
- massif – un profiler pentru heap, spune cât memory heap utilizează programul

Cel mai des folosit este memcheck, de aceea implicit valgrind îl folosește pe acesta, dacă se dorește folosirea altui utilitar acesta se va da ca parametru.

```
andrew@Goliath:~$ valgrind --tool=massif ./test

andrew@Goliath:~$ valgrind --tool=cachegrind ./test
```

Similar cu gdb, programul trebuie compilat cu flag-ul *-g* pentru a primit informații de depanare, astfel încât să se poată specifica linia exactă la care a apărut o problemă. Compilarea fără optimizări este și ea recomandă dacă este posibil, dacă acest lucru ar determina un timp de rulare prea mare se recomandă *-O1*. Folosirea unei optimizări de la *-O2* în sus poate produce erori de valori neinițializate care nu sunt corecte.

În continuare vom analiza un exemplu de rulare. Dacă în mod normal ați rula programul astfel:

```
andrew@Goliath:~$ ./test arg1 arg2
```

Pentru a rula în valgrind cu verificarea leak-urilor de memorie rulați următoarea comandă:

```
andrew@Goliath:~$ valgrind --leak-check=full ./test arg1 arg2
```

Utilitarul memcheck depistează următoarele tipuri de erori:

[1] http://en.wikipedia.org/wiki/Valgrind

- variabile neinițializate
- citiri/scrieri în zone de memorie eliberate
- citiri/scrieri în zone inaccesibile
- leak-uri de memorie

Exemple de output pentru anumite erori:

- citire invalidă

```
Invalid write of size 1
    at 0x804841E: main (example2.c:6)
Address 0x1BA3607A is 0 bytes after a block of size 10 alloc'd
    at 0x1B900DD0: malloc (vg_replace_malloc.c:131)
    by 0x804840F: main (example2.c:5)
```

- utilizare de variabile neinițializate

```
Conditional jump or move depends on uninitialised value(s)
   at 0x402DFA94: _IO_vfprintf (_itoa.h:49)
   by 0x402E8476: _IO_printf (printf.c:36)
   by 0x8048472: main (tests/manuel1.c:8)
```

- free-uri ilegale

```
Invalid free()
   at 0x4004FFDF: free (vg_clientmalloc.c:577)
   by 0x80484C7: main (tests/doublefree.c:10)
```

- suprapunerea adresei destinație cu cea sursă la copierea unor blocuri de memorie

```
Source and destination overlap in memcpy(0xbffff294, 0xbffff280, 21)
    at 0x40026CDC: memcpy (mc_replace_strmem.c:71)
    by 0x804865A: main (overlap.c:40)
```

- rezumat al leak-urilor de memorie

```
LEAK SUMMARY:
   definitely lost: 48 bytes in 3 blocks.
   indirectly lost: 32 bytes in 2 blocks.
     possibly lost: 96 bytes in 6 blocks.
   still reachable: 64 bytes in 4 blocks.
        suppressed: 0 bytes in 0 blocks.
```

Pentru mai multe detalii se va folosi parametrul *--leak-check=full*. Informațiile detaliate despre leak-uri vor arăta astfel:

```
8 bytes in 1 blocks are definitely lost in loss record 1 of 14
   at 0x........: malloc (vg_replace_malloc.c:...)
   by 0x........: mk (leak-tree.c:11)
   by 0x........: main (leak-tree.c:39)

88 (8 direct, 80 indirect) bytes in 1 blocks are definitely lost in
loss record 13 of 14
   at 0x........: malloc (vg_replace_malloc.c:...)
   by 0x........: mk (leak-tree.c:11)
   by 0x........: main (leak-tree.c:25)
```

Site-ul proiectului conține un manual de utilizare[1] în care se pot găsi mai multe informații despre erori[2] și opțiuni[3] pentru memcheck.

Valgrind rulează pe majoritatea platformelor Linux, există și portări neoficiale pentru NetBSD și FreeBSD.

14.9 Medii integrate de dezvoltare

Un mediu integrat de dezvoltare este construit pentru a spori productivitatea programatorilor prin oferirea unor componente strâns legate prin intermediul interfeței cu utilizatorul. Totuși, deoarece este vorba de un sistem complex, productivitatea sporită poate apărea după o perioadă lungă de adaptare și învățare a mediului.

De regulă, un mediu de dezvoltare (IDE) reprezintă un singur program în care se desfășoară tot procesul de dezvoltare: editare, compilare, depanare etc.

În mod obișnuit, un IDE este orientat pe un limbaj de programare specific, deși există și excepții.

14.9.1 Eclipse

Utilizând Java, având un comportament configurabil prin plugin-uri, mediul de dezvoltare Eclipse[4] poate fi utilizat pentru a dezvolta cod într-o mulțime de limbaje.

Sistemul de plugin-uri nu este doar un mecanism ce permite folosirea mai multor limbaje de programare, fiind posibilă și integrarea soluțiilor de versionare a codului (SVN), scrierea documentației LaTeX, testarea aplicațiilor de rețea etc.

Cu toate acestea, Eclipse are și câteva dezavantaje. Interfața grafică este puțin mai complicată. Fiind scris în Java, este absolut necesară prezența unei mașini virtuale Java în sistem pentru a putea fi folosit. Nu în ultimul rând, memoria consumată este destul de mare. De asemenea, fiind orientat pe proiecte, mediul de dezvoltare nu se pretează pentru programele mici (sub 300 de linii).

14.9.2 Anjuta

Spre deosebire de Eclipse, Anjuta[5] este un IDE pentru C și C++ pentru GNU/Linux. Scris pentru GTK și Gnome, programul oferă o serie de facilități precum managementul proiectelor, un depanator interactiv și un editor de cod ce permite syntax highlight și source browsing.

De asemenea, sunt posibile, print intermediul unor plugin-uri scrise în C (în viitor C++ și Python) diverse configurări. De exemplu, se poate schimba editorul de cod din Scintilla

[1] http://valgrind.org/docs/manual/manual.html
[2] http://valgrind.org/docs/manual/mc-manual.html#mc-manual.errormsgs
[3] http://valgrind.org/docs/manual/mc-manual.html#mc-manual.options
[4] http://www.eclipse.org
[5] http://projects.gnome.org/anjuta/index.shtml

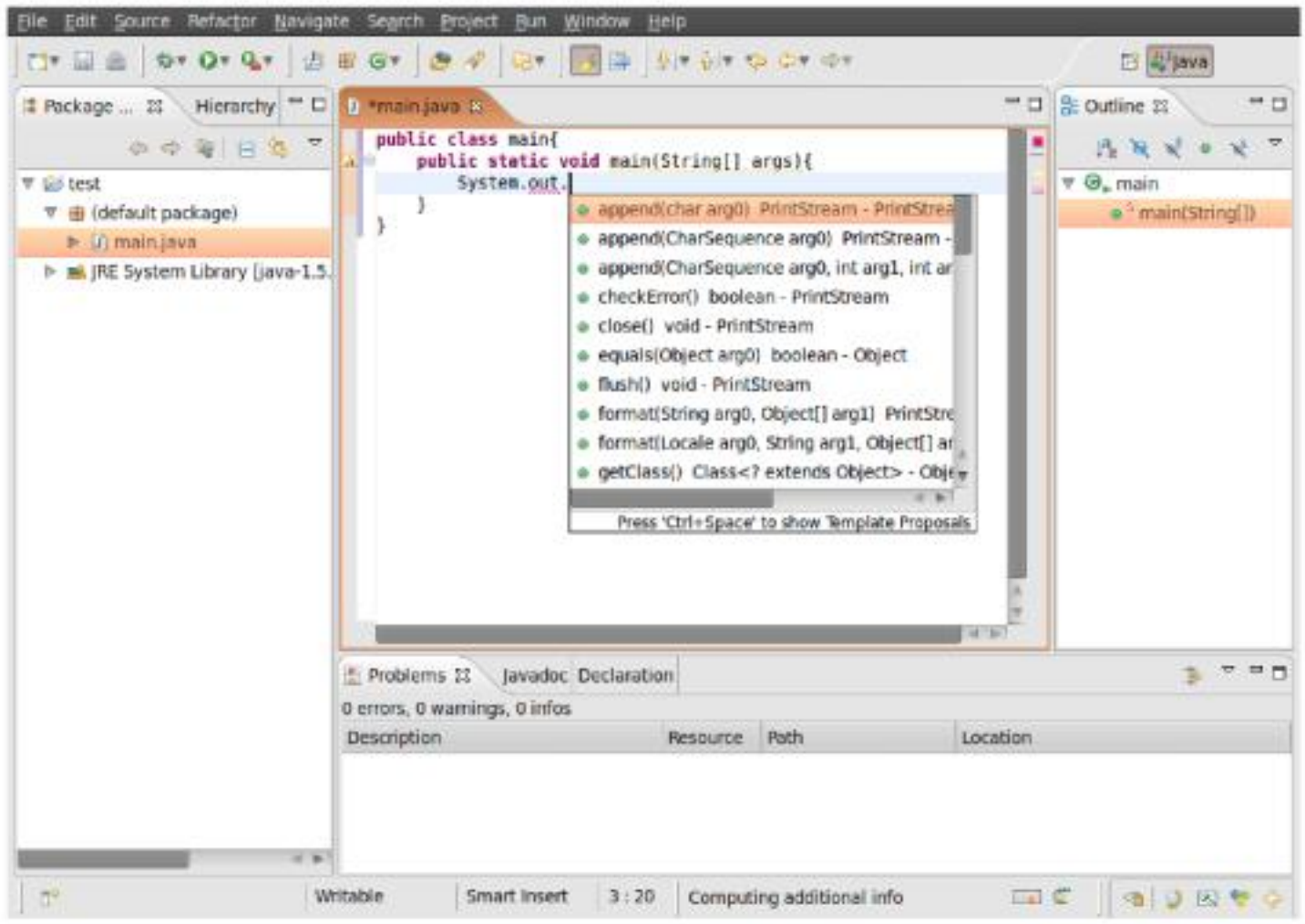


Figura 14.1: Eclipse

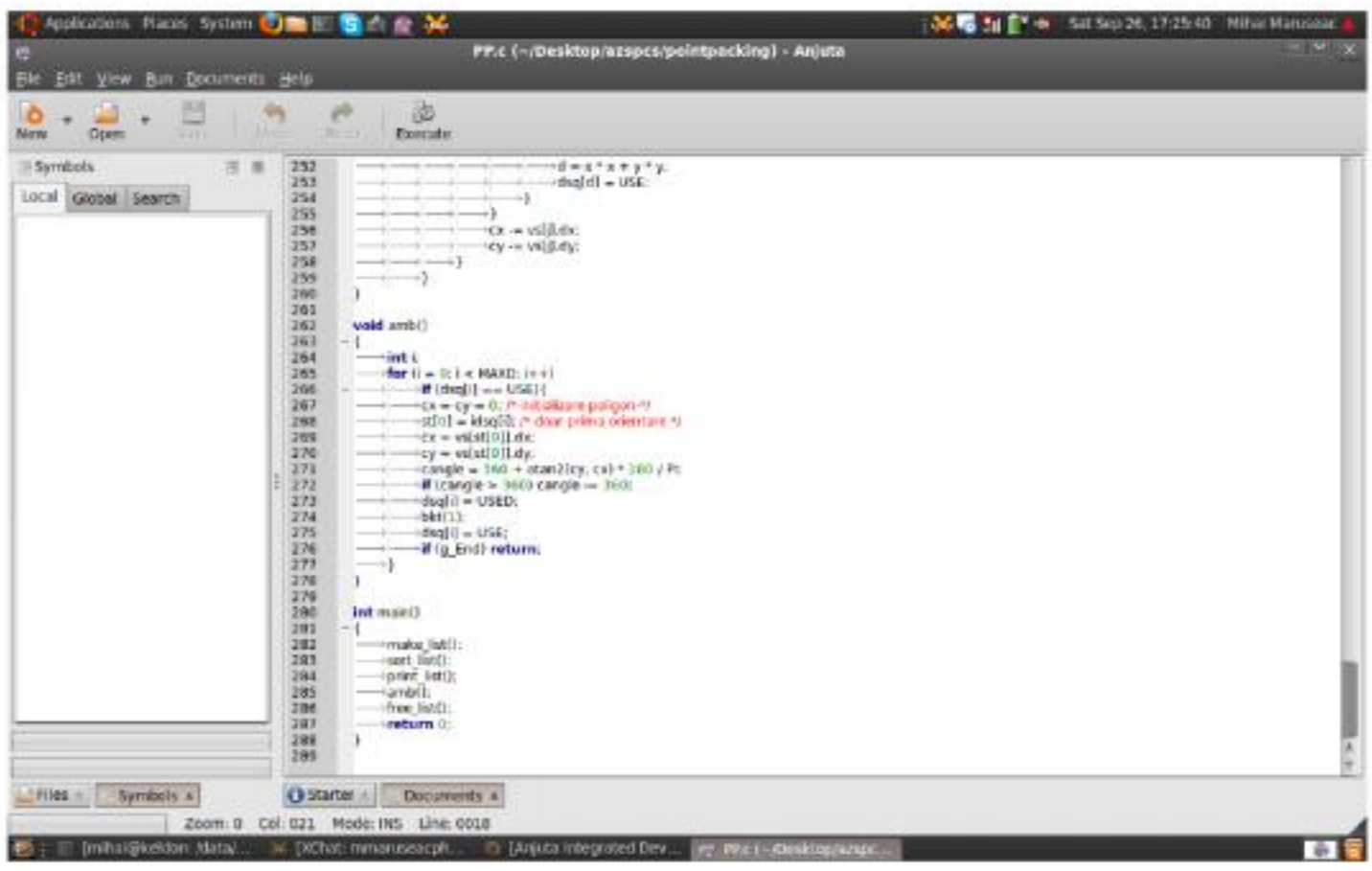


Figura 14.2: Anjuta

sau GtkSourceView în vim sau Emacs prin intermediul unui singur script scris de utilizator. Dacă Anjuta detectează mai multe plugin-uri pentru același scop, utilizatorul va fi rugat să selecteze unul din acestea, selecția fiindu-i memorată.

De asemenea, prin intermediul plugin-urilor se pot introduce facilități de source control (SVN). Mult mai important este faptul că managerul de proiecte nu reține informații despre proiect în alt fișier, astfel fiind posibilă dezvoltarea într-un grup mixt de utilizatori,

unii folosind Anjuta, alții folosind editoare clasice (vim, Emacs).

Pentru dezvoltarea aplicațiilor grafice Gtk, Anjuta oferă un plugin pentru interacțiunea cu Glade. Acceptând și limbajul C++, Anjuta oferă un generator de clase și un plugin ce va arăta ierarhia claselor din proiect. De asemenea, există plugin-uri pentru gdb, valgrind și gprof (ultimul pentru profiling-ul codului scris).

14.10 Managementul proiectelor software

O componentă importantă în dezvoltarea aplicațiilor software o reprezintă activitățile de gestiune, monitorizare și control a acestora. Managementul proiectelor software este legat de procesul de dezvoltare software și de modul în care o aplicație este proiectată, dezvoltată, testată și prezentată.

Activitățile de management al proiectelor software se referă la înțelegerea cerințelor aplicației, arhitectura și proiectarea acesteia, împărțirea de roluri și responsabilități în cadrul echipei, motivarea echipei, rezolvarea conflictelor, discuții, monitorizarea și controlul activităților, feedback, evaluarea rezultatelor, analiza calității. Domeniul managementului proiectelor sofware are o relevanță deosebită întrucât fiecare proiect software diferă de un altul. De asemenea, întrucât niciun limbaj de programare nu este perfect, procesele de dezvoltare trebuie să fie bine corelate cu limbajele, tehnologiile și framework-urile folosite.

14.10.1 Procese de dezvoltare software

Procesul de dezvoltare software reprezintă structura de activități folosită pentru dezvoltarea unui produs software.

Activitățile cuprinse în procesul de dezvoltare software sunt:

- **planificarea** – se referă extragerea cerințelor și stabilirea funcționalităților;
- **proiectarea** – se referă la stabilirea specificațiilor și a arhitecturii (module sau componente și interacțiunile între acestea);
- **implementare** – se referă la dezvoltarea efectivă a aplicației;
- **testarea** sau **verificarea** – însemnă verificarea respectării cerințelor software și a funcționării în condiții optime a aplicației;
- **documentarea** aplicației;
- **lansarea** aplicației pe piață sau livrarea acestuia clientului;
- **mentenanța** – se referă asigurarea funcționării corecte a aplicației în condițiile apariției de noi probleme și răspunsuri la solicitările clienților.

Există două tipuri de modele pentru implementarea procesului de dezvoltare software: modelul clasic (*waterfall*[1]) și modelul iterativ[2].

Modelul clasic presupune o dezvoltare pas cu pas a activităților procesului de management. O activitate nu va începe cât timp activitatea anterioară nu a fost definitivată. Acest model are avantajul unei dezvoltări riguroase, dar oferă o flexibilitate redusă, iar o echipă care lucrează la o anumită componentă va trebui să aștepte ca echipele anterioare să încheie componente proprie.

Modelul de dezvoltare iterativă presupune construirea unei părți reduse a proiectului software și apoi creșterea acesteia. Procesul iterativ este preferat în momentul în care nu se cunoaște foarte bine scopul urmărit. O subclasă a modelului iterativ o reprezintă dezvoltarea agilă (*agile software development*[3]). Dezvoltarea agilă folosește mecanismul de feedback ca principal mecanism pentru controlul și evoluția proiectului. Un alt exemplu de model iterativ este *Extreme Programming*[4] care presupune dezvoltarea de părți foarte mici ale unui proiect (de obicei în echipe mici) și apoi integrarea acestora în proiect.

14.10.2 Aplicații web pentru managementul proiectelor software

Există un set divers de aplicații de gestiune a proiectelor software. În cadrul acestora, o categorie aparte o reprezintă aplicațiile web pentru managementul proiectelor software. Aceste aplicații oferă, în general, un set integrat de componente printre care:

- wiki – un wiki este o aplicație web care permite editarea colaborativă facilă de conținut; celebra enciclopedie online Wikipedia folosește tehnologia wiki pentru dezvoltare colaborativă;
- un repository (vezi secțiunea 14.4) pentru gestiunea codului sursă;
- sistem de tichete pentru raportarea problemelor (*bug-urilor*) și pentru cerințe de noi funcțialități;
- roadmap și milestone-uri pentru planificarea și contorizarea activităților.

Trac[5] este o aplicației minimalistă pentru gestiunea proiectelor software. Trac folosește Subversion ca aplicație pentru gestiunea codului. Un aspect pozitiv al Trac este numărul mare de plugin-uri disponibile[6]. Trac este scris în Python și este folosit de un număr mare de organizații. **Redmine**[7] este o aplicație web pentru gestiunea proiectelor software scrisă folosind Ruby on Rails[8]. Redmine a fost influențat puternic de Trac. Redmine oferă în plus față de Trac o diversitate de utilitare de gestiune a codului (Subversion, Git, Mercurial, Bazaar, Darcs), gestiunea de proiecte multiple în cadrul aceleiași instanțe și folosirea de diagrame Gantt și de calendare.

[1] http://en.wikipedia.org/wiki/Waterfall_model
[2] http://en.wikipedia.org/wiki/Iterative_development
[3] http://en.wikipedia.org/wiki/Agile_software_development
[4] http://en.wikipedia.org/wiki/Extreme_Programming
[5] http://trac.edgewall.org/
[6] http://trac-hacks.org/
[7] http://www.redmine.org/
[8] http://www.rubyonrails.org/

14.10.3 Resurse online pentru dezvoltarea proiectelor

Pentru cei care nu doresc instalarea unei instanțe de Trac sau Redmine pentru dezvoltarea și managementul proiectelor sofware, se po folosi resurse online. În general, accesul la acestea este gratis. Oricine poate crea un cont asociat unui proiect și poate folosi resursele puse la dispoziție pentru dezvoltarea acestuia.

Aplicațiipe ce stau la baza acestor site-uri sunt denumite `software forge-uri`[1]. Majoritatea forge-urilor pot fi instalate separat. Principalele site-uri care oferă servicii de găzduire a proiectelor software sunt SourceForge[2], GNU Savannah[3], Google code[4], BerliOS[5], CodePlex[6].

14.11 Resurse de documentare pentru dezvoltator

Mai mult decât în orice alt domeniu, în IT documentația este ușor de obținut, de la format electronic până la cărți tradiționale, din surse oficiale sau contribuții ale altor persoane.

14.11.1 Documentație oficială

Pachetele software cunoscute vin, în general, cu documentație detaliată. Aceasta este accesibilă, în cazul programelor simple și a bibliotecilor, prin pagini de manual (**`man`**).

În cazul programelor GNU[7], manualul nu conține decât un sumar al comenzilor. Pentru informații mai detaliate se folosește **`info`**. Există alternative la **`info`**, printre care cititorul integrat din Emacs, sau **`pinfo`**[8].

Alte suite software complexe, în general limbaje de programare, posedă utilitare personalizate pentru vizualizarea documentației. Câteva exemple sunt:

- `pythondoc` pentru documentarea limbajului Python[9];
- `perldoc` pentru documentarea limbajului Perl[10];
- `texdoc` pentru documentarea limbajului TEX[11] și a suitei de macro-uri pentru el, LATEX[12].

[1] http://en.wikipedia.org/wiki/Forge_(software)
[2] http://sourceforge.net/
[3] http://savannah.gnu.org/
[4] http://code.google.com/
[5] http://www.berlios.de/
[6] http://www.codeplex.com/
[7] http://www.gnu.org/
[8] http://pinfo.sourceforge.net/
[9] http://www.python.org/
[10] http://www.perl.org
[11] http://tug.org/
[12] http://www.latex-project.org/

14.11.2 Documentarea programelor proprii

Orice program trebuie documentat, chiar dacă nu este intenția explicită a programatorului ca sursa să fie citită de altcineva. Deși, în general, comentarii sumare la începutul fiecărei funcții sunt suficiente, există programe care permit comentarea structurată și generarea de documentație în varii formate (HTML, pagini de manual, LaTeX). Este, însă, important de menționat că documentația generată este adresată *dezvoltatorilor*, și nu utilizatorilor.

Pentru ca un astfel de program să funcționeze corect, este nevoie de o metodă de a delimita comentariile uzuale de cele care contribuie la generarea automată a documentației. Două exemple de formate speciale pentru limbajele din familia C/C++ sunt începerea comentariilor cu `/**` sau cu `///`.

Un astfel de program este **Doxygen**[1]. Este open-source, are suport pentru mai multe limbaje și generează documentație în formate variate. Este, în general, prima alegere pentru proiecte open-source care aleg să genereze documentația automat. Un exemplu excelent de documentație generată de Doxygen este cea a proiectului Pidgin[2].

Un exemplu de comentariu compatibil cu Doxygen, pentru o funcție C:

```
1  /**
2   * Executa comenzile corespunzatoare unui arbore.
3   *
4   * @param c comanda compusa care trebuie executata
5   * @param pi descriptorul care trebuie duplicat pentru intrare
6   * @param po descriptorul care trebuie duplicat pentru iesire
7   * @param w este 1 daca trebuie sa asteptam comanda sa se termine
8   *
9   * @return Codul de iesire al ultimului proces executat, sau pidul
10  * procesului nou creat daca acesta nu a fost asteptat. Daca comanda
11  * executata a fost quit sau exit, se intoarce valoarea speciala
12  * QUITTING.
13  *
14  * @remarks pi si po sunt folosite atunci cand comanda face parte
15  * dintr-un pipe. Pentru a nu le folosi, le setam la -1.
16  */
17 int exec_command(const command_t *c, int pi, int po, int w)
18 {
19   [...]
```

Limbajul Java are un instrument propriu pentru generarea documentației de API, numit `Javadoc`[3]. Toată documentația platformei Java[4] este generată utilizând `Javadoc`, iar formatul Doxygen este voit compatibil cu cel Javadoc pentru a fi o alternativă facilă.

C# oferă o alternativă interesantă, folosind comentarii în format XML și marcate cu `///`. Deși marcarea câmpurilor este mai dificilă, mediul Visual Studio oferă suport excelent pentru acest tip de comentarii.

---

[1] http://www.doxygen.org/
[2] http://developer.pidgin.im/doxygen/dev/html/classes.html
[3] http://java.sun.com/j2se/javadoc/
[4] http://java.sun.com/j2se/1.4.2/docs/api/

14.11.3 Cărți și tutoriale

Un tutorial este o "scurtătură", o referință rapidă, dar incompletă a unei tehnologii sau limbaj. O astfel de resursă poate substitui temporar documentația propriu-zisă, în faza de familiarizare, dar detaliile care lipsesc (intenționat, pentru a asimila mai ușor lucrurile de bază) sunt adesea esențiale pentru o utilizare eficientă a conceptului în cauză.

Deși documentația "la obiect" este accesibilă sub formă electronică, știința calculatoarelor nu duce lipsă de cărți în format clasic. Acestea nu sunt nicidecum redundante, pentru că deseori conțin sugestii, subtilități și practici foarte valoroase.

O categorie importantă de cărți sunt cele care își propun să detalieze un domeniu întreg, independent de implementare. Acestea conțin elemente teoretice avansate, dar și studii de caz relevante. În general, fiecare domeniu are o asemenea carte, recunoscută și respectată de persoanele în temă. Pentru că nu sunt "încuiate" în studiul unei implementări particulare, aceste cărți sunt preferate în domeniul academic.

Exemple de asemenea cărți și domeniile[1] lor asociate sunt:

- sisteme de operare: Silberschatz, Galvin, Gagne, *Operating System Concepts* [26]; Tanenbaum – *Modern Operating Systems* [32];
- rețele de calculatoare: Tanenbaum – *Computer Networks* [30];
- arhitectura sistemelor: Hennessy, Patterson – *Computer Architecture: A Quantitative Approach* [9];
- compilatoare: Aho, Sethi, Ullman – *Compilers: Principles, Techniques, and Tools* [2].

Nu în ultimul rând, profesorul Donald Knuth scrie o veritabilă monografie a programării calculatoarelor, numită *The Art of Computer Programming* [14]. Începută în 1962, cartea este programată să aibă 7 volume, dintre care doar primele 3 au apărut complet.

Alte cărți se referă exclusiv la programare și descriu în detaliu fie un limbaj, fie un set de utilitare sau un mediu întreg de programare. Spre deosebire de categoria anterioară, aceste cărți se adresează unui domeniu mai îngust de probleme, dar adoptă o abordare *hands-on*, conținând frecvent exemple de cod.

Lucrări celebre în acest sens sunt: Kernighan, Ritchie – *The C Programming Language* [13], Wall, Christiansen, Orwant – *Programming Perl*[2] [33], Kernighan, Pike – *The Practice of Programming* [12], sau Eckel – *Thinking in Java* [5].

În cele din urmă, există cărți care descriu în detaliu un program sau o parte a unui sistem complex. Acestea se remarcă printr-un grad înalt de specificitate și limbaj tehnic, dar oferă o detalii fine despre subiectul abordat. Asemenea lucrări sunt apreciate în special în industrie, de persoane care configurează sau modifică programe ca parte a slujbei lor.

În această categorie se încadrează, de exemplu, cărțile lui Robert Love despre kernelul Linux, *Learning the vi and Vim Editors* de Robbins, Hannah și Lamb, sau *DNS and BIND* de Liu și Albitz.

---

[1] Desigur, domeniile se întrepătrund. Un inginer nu își permite să se limiteze la un singur aspect al profesiei lui.

[2] cunoscută cu afecțiune în comunitatea Perl drept *The Camel Book*

Departe de domeniul tehnic, dar relevante pentru industria IT, sunt lucrările cu tentă ideologică sau cele care observă și compară tendințe. Acestea tind să se refere la procesul producerii software-ului, fie el proprietar sau liber. Titluri notabile sunt Brooks – *The Mythical Man-Month*, Raymond – *The Cathedral and the Bazaar* și Fogel – *Producing Open Source Software*[1].

Ca o notă finală asupra cărților, este esențial a realiza că ele dezvoltă competențe distincte față de citirea documentației exclusiv tehnice. Cele două tipuri de resurse se folosesc împreună – ele se completează, nu se substituie.

14.11.4 Documentație din Internet

Pe lângă documentația oficială, există multe alte surse, fiecare cu specificul ei. Gradul de detaliu și interacțiunea dintre persoane variază funcție de mijlocul de comunicație.

IRC

IRC (*Internet Relay Chat*) este un protocol de comunicație multidirecțional în timp real. Clienți uzuali pentru acest mediu sunt Pidgin[2], XChat[3], sau irssi[4].

Deși protocolul însuși, în special datorită utilizării programului mIRC, are o reputație îndoielnică, există mult pachete software și comunități care oferă suport informal folosind IRC. Este utilă o asemenea abordare când se dorește un răspuns în timp real, la obiect, dar din care de multe ori lipsesc detalii care nu sunt imediat relevante.

Majoritatea proiectelor care folosesc IRC drept mijloc de comunicare sunt găzduite de rețeaua Freenode[5]. Dezvoltatorii se lansează frecvent în discuții informale, fără legătură cu subiectul, de aceea protocolul are un *signal-to-noise ratio* destul de mic.

Usenet

Usenet este o rețea care precede cronologic World Wide Web-ul. Canalul de comunicație este tot *one-to-many*, dar, spre deosebire de IRC, această metodă este asincronă, asemănându-se cu email-ul.

Datorită vârstei "venerabile" a și a lipsei sale relative de popularitate, Usenet este folosit de unii utilizatori mai în vârstă, dar care au cunoștințe tehnice excepționale. Usenet conține discuții tehnice detaliate, dar și atacuri personale și *flame-wars*.

În ultimii ani, ISP-urile au încetat să mai ofere servere de Usenet, dar o metodă excelentă de a citi și scrie mesaje este arhiva Google Groups[6]. Un grup de o notorietate legendară, atât pentru discuțiile referitoare la limbajul C, cât și pentru flame-urile numeroase, este `comp.lang.c`.

---

[1] http://producingoss.com/
[2] http://pidgin.im/
[3] http://xchat.org
[4] http://irssi.org
[5] http://freenode.net/
[6] http://groups.google.com/groups/dir?sel=gtype%3D0

Mai mult decât orice altă sursă de informație, Usenet este neiertător cu persoanele care fac greșeli. Se recomandă citirea câtorva tread-uri de discuție înainte de a porni o discuție nouă[1].

Google

O ultimă sursă de informație este a folosi un motor de căutare. Avantajul este viteza cu care acesta caută prin majoritatea surselor menționate mai sus, inclusiv discuții Usenet sau log-urile canalelor de IRC. Pe de altă parte, este uneori dificilă alegerea cuvintelor potrivite pentru exprimarea unei probleme.

14.12 Studiu de caz

14.12.1 Microsoft Visual Studio

Microsoft Visual C++ (MSVC) este un IDE(Integrated Development Enviroment) făcut de Microsoft pentru limbajele de programare C, C++ și C++/CLI. Acesta conține utilitare de depanare și dezvoltare a codului scris în C++.

Visual C++, componenta care se ocupă de codul C/C++, este compus din mai multe elemente:

- compilatorul Visual C++, conține optimizări pentru platformele x86, x64 și Itanium
- bibliotecile Visual C++, cum ar fi ATL(Active Template Library), MFC(Microsoft Foundation Class), CRT(C RunTime Library) și biblioteca standard C
- mediul de dezvoltare Visual C++. Compilatorul și bibliotecile pot fi accesate și prin interfața în linie de comandă, mediul de dezvoltare oferă posibilități de dezvoltare și depanare mai facilă a proiectelor

Există mai multe metode de a crea un proiect în Visual C++:

- se poate folosi un șablon, cum ar fi *Console Application Template*, pentru a realiza proiecte simple
- se poate folosi și un wizard pentru crearea unei soluții, o soluție poate să conțină mai multe proiecte
- se poate crea un fișier text simplu salvat cu extensia *.cpp*, iar acesta să fie inclus într-un proiect gol de tipul *Win32 Application*

Folosind wizard-ul avem acces la o interfață ce permite să se creeze un proiect, să se modeleze un proiect după un șablon și să se generezeze fișiere sursă și directoare pentru aplicație. Visual C++ lucreaza cu framework-uri de aplicații și biblioteci pentru a crea schelete de programe peste care se poate lucra ulterior.

Mai multe detalii despre proiecte puteți găsi pe site-ul MSDN[2].

---

[1] *Lurk before you leap.*

[2] http://msdn.microsoft.com/en-us/library/6765tta0.aspx

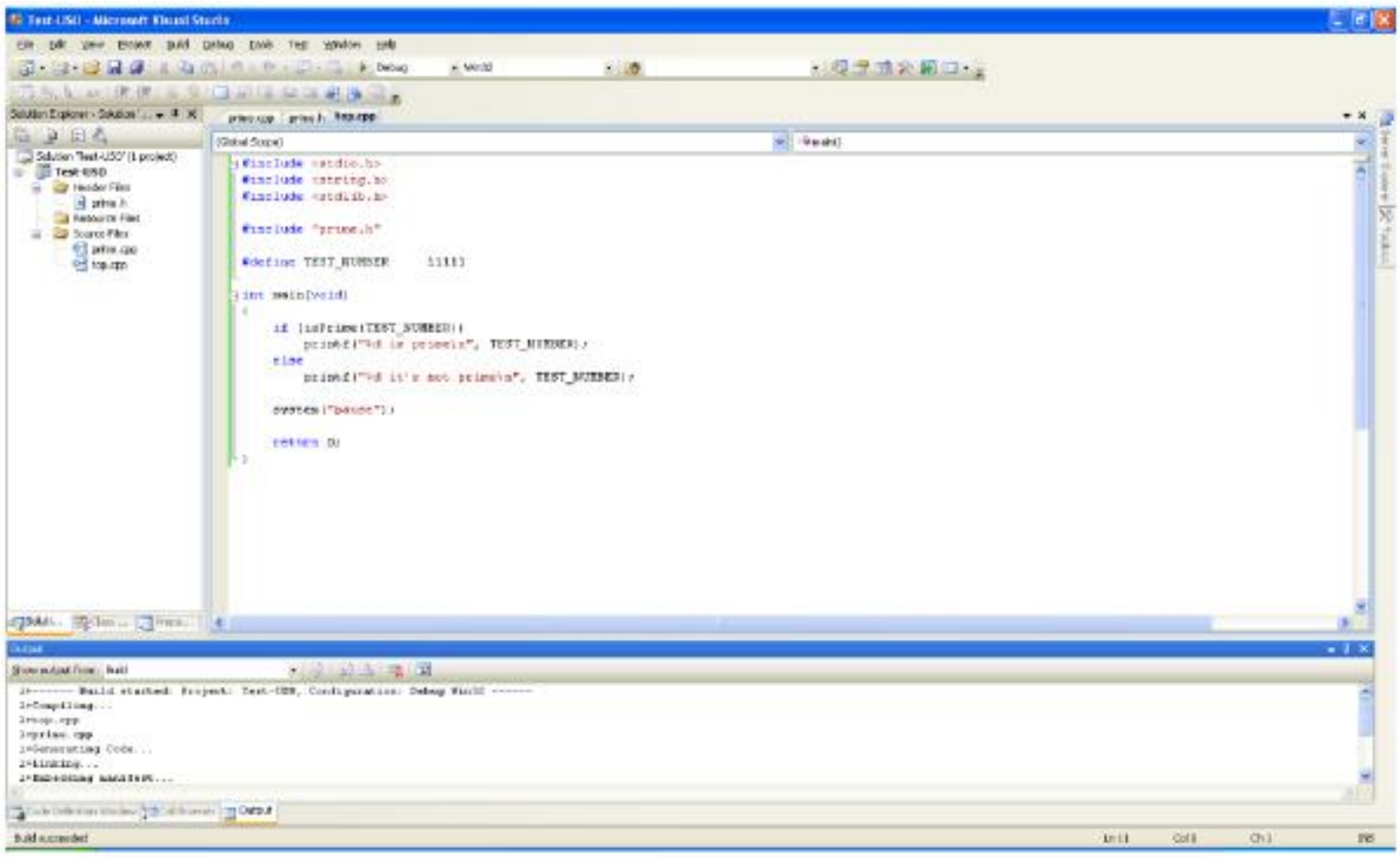


Figura 14.3: Proiect în Visual C++

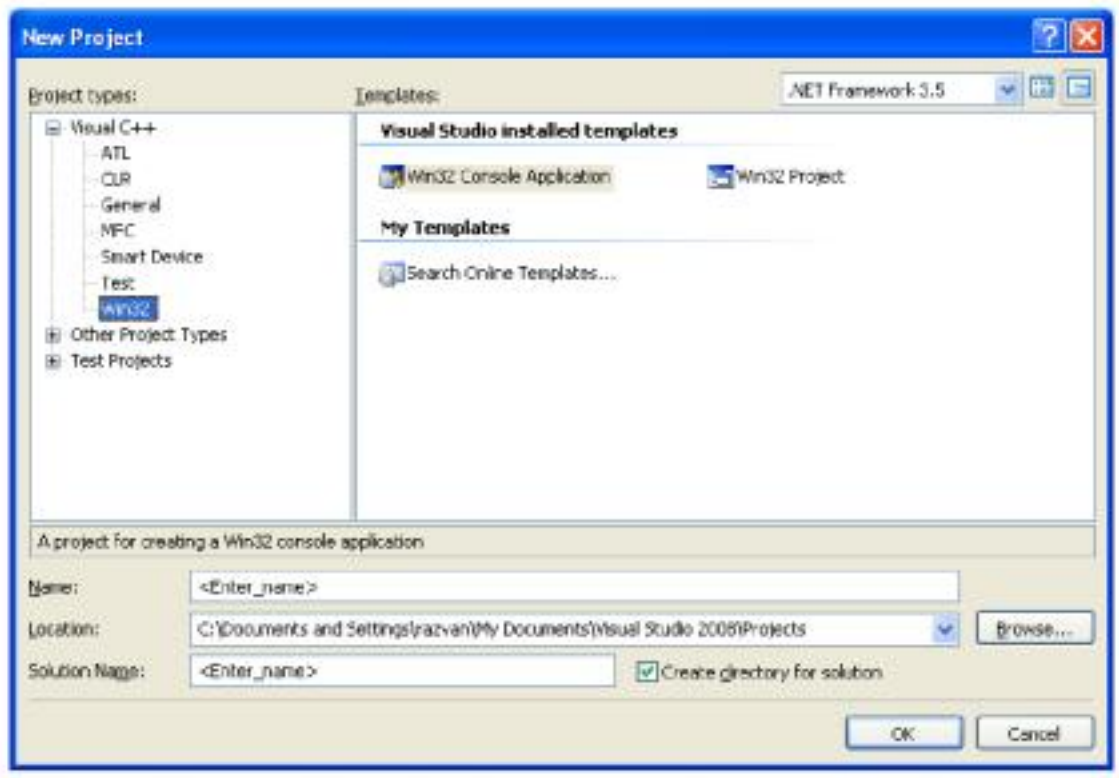


Figura 14.4: Creare proiect în Visual C++

Cuvinte cheie

- coding style
- indent, astyle
- editor
- vim
- modul insert, command, visual
- controlul versiunii
- subversion
- git
- ctags
- fișiere index (tag)
- analiză statică
- splint

- GNU Autotools
- strace
- .data, .bss, .rodata
- stivă
- heap
- apeluri de sistem
- ldd
- depanare
- GDB
- ddd
- Valgrind
- memory leak
- IDE
- Eclipse
- Anjuta
- waterfall
- model iterativ
- Trac
- Redmine
- man
- info
- Doxygen
- Javadoc
- IRC – Internet Relay Chat
- Usenet
- Microsoft Visual Studio

Întrebări

1. Care din următoarele este un editor?
   - ❑ Vim
   - ❑ astyle
   - ❑ ctags
   - ❑ gdb
2. Care din următoarele aplicații este folosită pentru depanarea la runtime a unui program?
   - ❑ Valgrind
   - ❑ Trac
   - ❑ indent
   - ❑ git
3. Care din următoarele operații NU este o operație validă la folosirea unui sistem de gestionare a versiunii?
   - ❑ commit
   - ❑ update
   - ❑ checkout/clone
   - ❑ boot
4. Care din următoarele NU se referă la recomandări din cadrul unor convenții de codare?

- ❑ nume sugestive pentru variabile
- ❑ funcții cu dimensiune rezonabilă
- ❑ folosirea de spații și linii libere
- ❑ licențierea codului sub GPL

5. Care din următoarele NU este un IDE?
   - ❑ Anjuta
   - ❑ Eclipse
   - ❑ Microsoft Visual Studio
   - ❑ OpenOffice
6. Care din următoarele utilitare este folosit în conjuncție cu Vim pentru parcurgerea facilă a codului?
   - ❑ ctags
   - ❑ splint
   - ❑ indent
   - ❑ gdb
7. Care din următoarele NU este un mod de folosire pentru editorul Vim?
   - ❑ `visual`
   - ❑ `command`
   - ❑ `insert`
   - ❑ `graphic`
8. Care din următoarele comenzi este folosită, în Vim, pentru salvarea unui fișier?
   - ❑ `:w`
   - ❑ `:q`
   - ❑ `:s`
   - ❑ `:help`
9. Care din următoarele este un sistem pentru controlul versiunii?
   - ❑ Git
   - ❑ Vim
   - ❑ Eclipse
   - ❑ ddd
10. Care din următoarele este o aplicație web pentru gestiunea proiectelor software?
    - ❑ Trac
    - ❑ GTK

- ddd
- Waterfall

Capitolul 15

Viața în Linux

See, you not only have to be a good coder to create a system like Linux, you have to be a sneaky bastard too ;-)

Linus Torvalds

Ce se învață din acest capitol?

- Cu ce se ascultă muzică în Linux
- Cum se vede un film în Linux
- Cum se scrie un CD/DVD în Linux
- Cum se foloseste Y!M în Linux
- Cum se descarcă conținut BitTorrent în Linux
- Cum se folosește imprimanta în Linux
- Ce jocuri 3D și 2D există în Linux
- Cum se folosesc două monitoare în Linux
- Cum se editează documente Word/Excel în Linux

15.1 Muzica în Linux

Un player bun de muzică trebuie să asigure, pe lângă posibilitatea de a reda fișierele audio, managementul acestora, salvarea playlist-urilor, posibilitatea de "minimize to tray" și, eventual, posibilitatea de audio-scrobbling (trimiterea informațiilor despre muzica ascultată unor site-uri de profil, precum last.fm[1]). Un utilizator mai pretențios ar cere și management-ul fișierelor de pe Audio-CD-uri, iPod-uri etc.

Din fericire, majoritatea acestor condiții sunt respectate de player-ele audio din Linux, fără a fi necesară instalarea unor plugin-uri suplimentare (acestea fiind instalate automat în momentul instalării aplicației).

[1] http://www.last.fm/

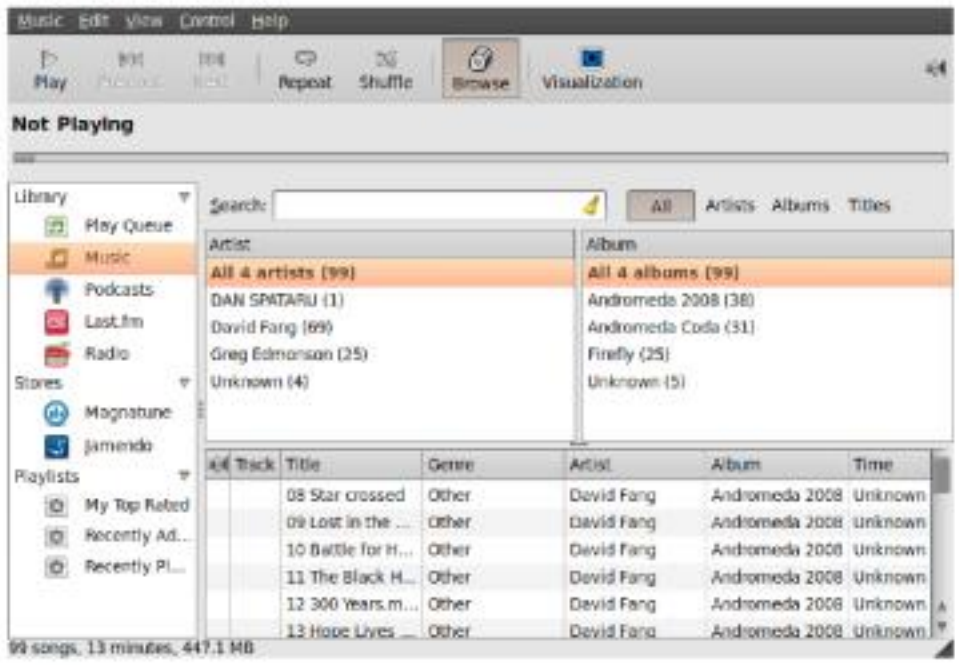


Figura 15.1: Rhythmbox

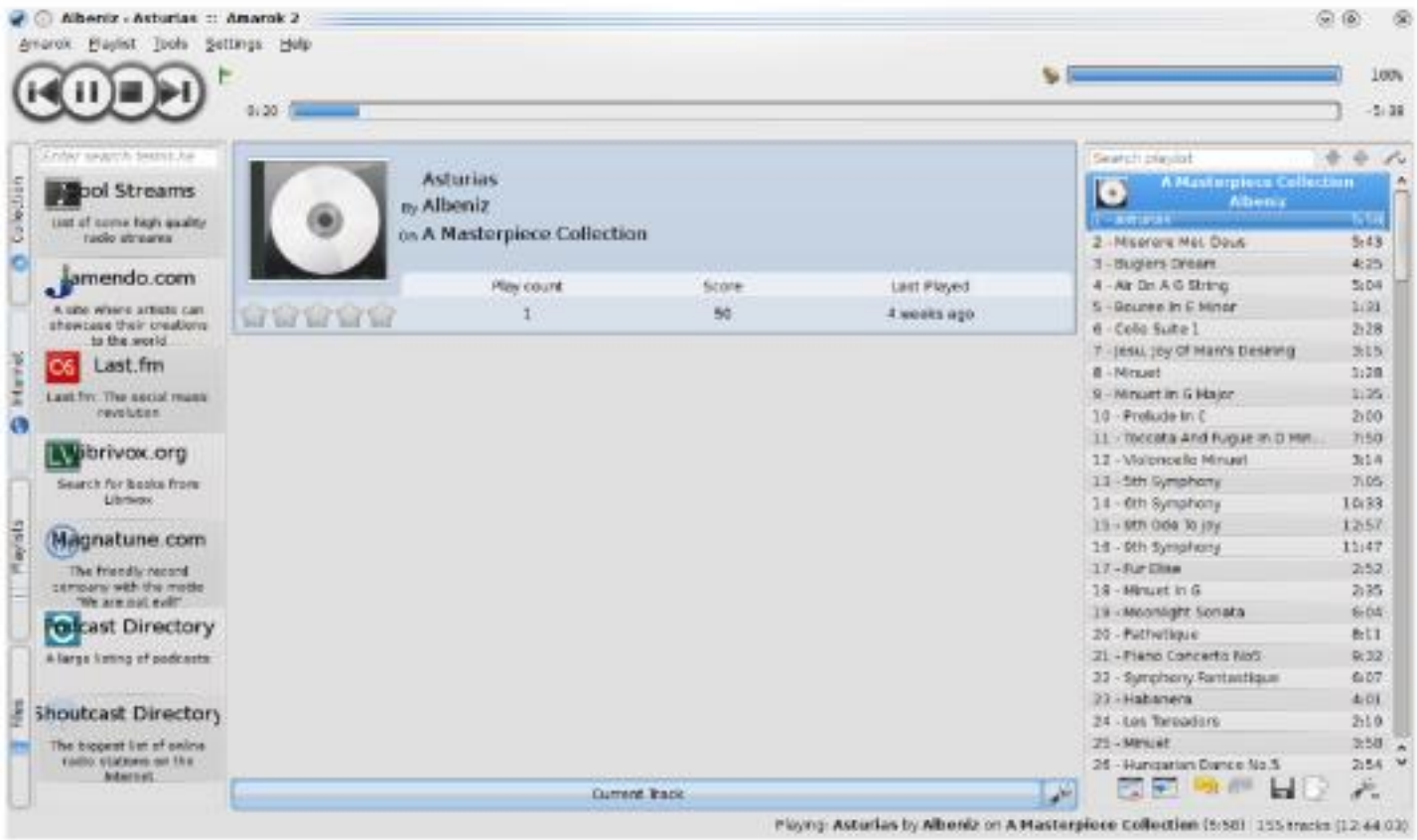


Figura 15.2: Amarok

Sistemele de operare Ubuntu au preinstalat **Rhythmbox**. Nu este doar un player audio ci și o aplicatie pentru managementul fișierelor audio, inspirată din aplicația iTunes de la Apple. Echivalent, în mediile Kubuntu este preinstalat **Amarok**.

Dacă aplicațiile preinstalate nu corespund preferințelor lui, utilizatorul poate oricând să instaleze altele, precum:

- **Banshee**[1] – o aplicație similară Rhythmbox, bazată pe mono
- **Listen**[2] – o aplicație minimală scrisă în Python
- **Audacious**[3] – un player similar aplicației Winamp din Windows

[1] http://banshee-project.org/
[2] http://www.listen-project.org/
[3] http://audacious-media-player.org

Figura 15.3: Audacious

- **Exaile**[1] – o aplicație similară Amarok pentru GNOME

O aplicație interesantă este Music Player Daemon[2]. Așa cum îi spune și numele, programul rulează ca daemon, dar este controlat, fie local, fie prin rețea, de un client. Există zeci de astfel de clienți, pentru diverse platforme, scriși în varii limbaje de programare și cu interfețe grafice sau în mod text.

15.2 Filme în Linux

Vizualizarea de filme este deseori mai facilă în Linux decât în Windows, pentru că programele care există sunt toate gratuite în adevăratul sens al cuvântului, adică nu vin cu alt software de proveniență îndoielnică. Mai mult, în general instalarea unui video player pe Linux implică automat instalarea codecurilor asociate, dacă ele nu sunt deja instalate.

VLC este un player produs de proiectul VideoLAN. Acesta are avantajul de a avea codec-uri integrate, ceaa ce reduce pachetele adiționale care trebuiesc instalate cu el. Pe sistemele Debian, pachetul asociat se numește `vlc`.

O alternativă la VLC este `mplayer`, cu o interfață mai minimală, dar la fel de bogat în facilități. Utilizatorii `mplayer` pot alege între mai multe variante de interfețe, de la mod text până la GUI-uri sofisticate și interfețe native (pentru Mac OS X, de exemplu). `mplayer` are, de asemenea, posibilitatea de a reda filme folosind mai multe drivere de output, folosind parametrul `-vo` în linie de comandă. Astfel, dacă pachetele potrivite sunt instalate, un utilizator nu are, de exemplu, nevoie de interfață grafică pentru a vizualiza un fișier. `mplayer` poate afișa direct în framebuffer folosind biblioteca `directfb` sau în mod text folosind `aalib`[3].

Ambele programe menționate folosesc preponderent biblioteca `libavcodec`, parte a proiectului FFmpeg. Pentru unele formate specifice Windows, se pot instala drivere

[1] http://www.exaile.org

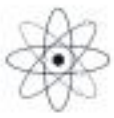
[2] http://mpd.wikia.com/wiki/Music_Player_Daemon_Wiki

[3] `aalib` folosește text pentru a reda imagini cât mai bine, deci poate fi folosit peste conexiuni text-only.

proprietare, numite generic `w32codecs`. Nu în ultimul rând, ambele pachete sunt portabile, deci pot fi rulate cu ușurință și pe Windows.

15.3 Scrierea unui CD/DVD în Linux

În mod clasic, scrierea de CD-uri/DVD-uri în Linux se realizează cu ajutorul utilitarului **cdrecord**. În ultimele versiuni, aplicația a fost denumită **wodim**. Aplicațiile grafice folosite uzual pentru scrierea de CD-uri (GnomeBaker, Brasero, K3b) folosesc în spate utilitarul **cdrecord**.



Pentru a folosi **wodim**/**cdrecord** este nevoie de o imagine de CD (fișier `.iso`. Pentru a obține o astfel de imagine se folosește utilitarul **`genisoimage`** (fostul **`mkisofs`**):

```
1  razvan@valhalla:~/projects$ ls library/
2  eLiberatica_OReilly_books.odt  eLiberatica_OReilly_lista_carti.gnumeric
3  eLiberatica_OReilly_carti.odt  eLiberatica_OReilly_lista_carti.ods
4
5  razvan@valhalla:~/projects$ genisoimage -o library.iso library/
6  Using ELIBE000.ODT;1 for  /eLiberatica_OReilly_carti.odt (
   eLiberatica_OReilly_books.odt)
7  Total translation table size: 0
8  [...]
9
10 razvan@valhalla:~/projects$ file library.iso
11 library.iso: ISO 9660 CD-ROM filesystem data 'CDROM
```

Scrierea CD-ului se face cu o comandă ca cea de mai jos:

```
1  razvan@valhalla:~/projects$ cdrecord dev=/dev/hda speed=8 library.iso
```

Imaginea `.iso` poate fi montată și accesată în sistemul local de fișiere, similar utilitarului Daemon Tools[1] din Windows:

```
1  razvan@valhalla:~/projects$ mkdir tmp_mount
2
3  razvan@valhalla:~/projects$ sudo mount -o loop -t iso9660 library.iso
   tmp_mount/
4
5  razvan@valhalla:~/projects$ ls tmp_mount/
6  elibe000.odt  eliberat.gnu  eliberat.ods  eliberat.odt
```

Se observă că, în urma obținerii imaginii, fișierele au fost redenumite în formatul **8.3**[2].

În GNOME, scrierea CD-urilor se poate realiza cu ajutorul utilitarului **GnomeBaker**. Pentru scrierea unui CD, se pornește utilitarul, se selectează opțiunea `Data CD` și apoi se adaugă directoarele de scris pe CD în arborescența din partea stânga jos, așa cum reiese din figura 15.4. În momentul în care toate fișierele au fost adăugate se folosește butonul `Burn`.

În KDE, scrierea CD-urile se realizează cu ajutorul utilitarului **K3b** (vezi figura 15.5). Modul de utilizare este foarte apropiat de cel al GnomeBaker.

[1] http://www.daemon-tools.cc/eng/home
[2] http://en.wikipedia.org/wiki/8.3_filename

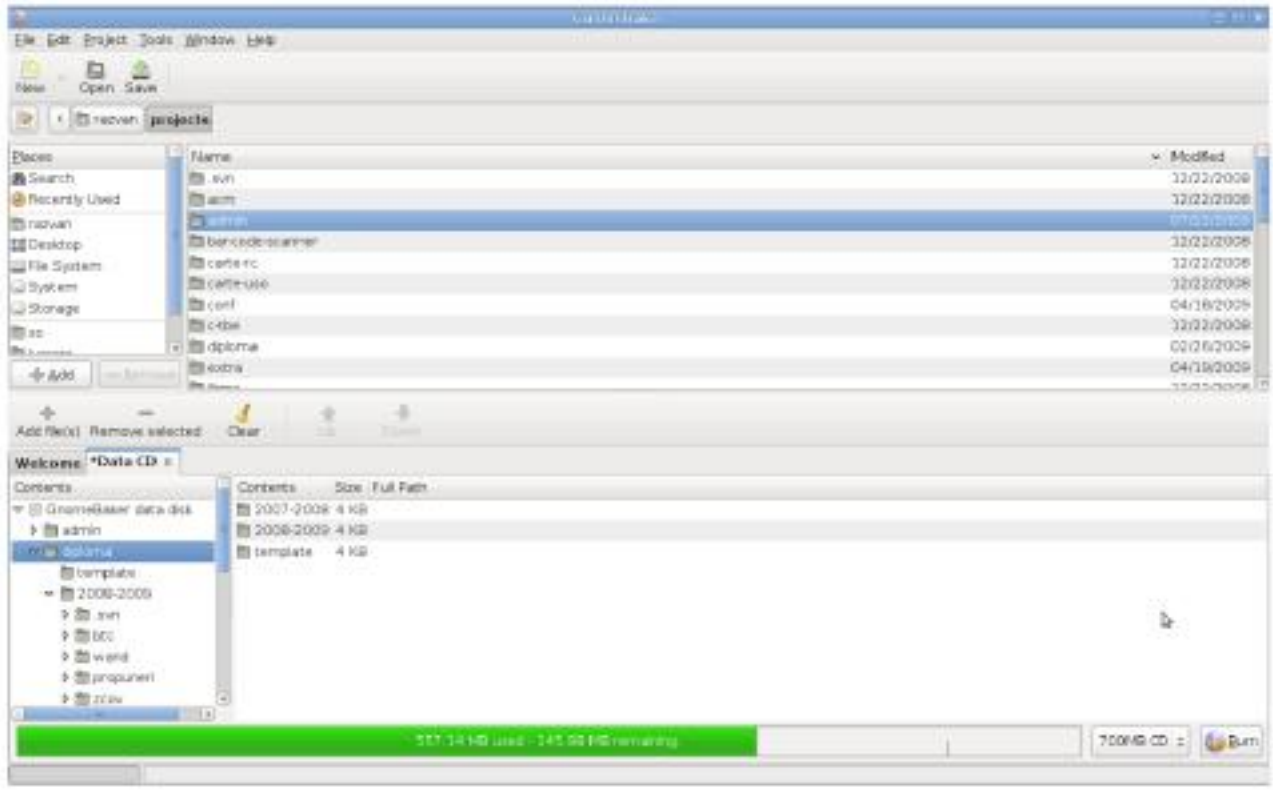


Figura 15.4: Scrierea unui CD folosind GnomeBaker

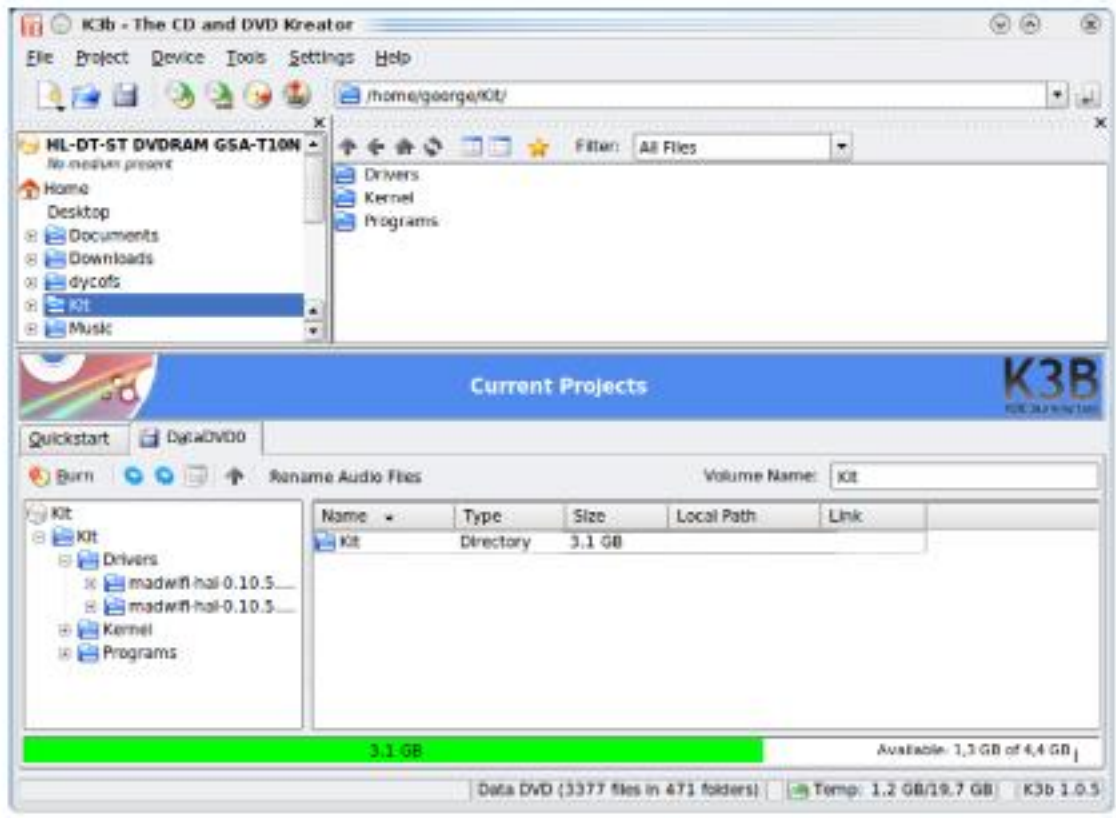


Figura 15.5: Scrierea unui CD folosind K3b

15.4 Messenger în Linux

Pentru utilizatorii de *Instant Messaging*, Linux are, tradițional, o alternativă ușor diferită, dar mult mai flexibilă și mai utilizabilă.

Acest program se numește **Pidgin**. Inițial numit Gaim, Pidgin are avantajul de a fi multi-protocol. Folosind aceeași interfață și o singură instanță a programului, un utilizator poate avea acces la conturile de Yahoo Messenger, MSN/Windows Live, Google Talk, AIM etc. Mai mult, este posibilă folosirea mai multor conturi din aceeași rețea simultan, fără a fi nevoie de instalarea unui patch adițional.

Central conceptului de conturi multiple din Pidgin este fereastra "*Accounts*". Aceasta poate fi accesată din meniul `Accounts→Manage Accounts`. Adăugarea unui cont

de Yahoo este cât se poate de facilă; ea nu necestă configurări adiționale. Google Talk necesită specificarea unui domeniu, pentru că se bazează pe un protocol descris numit XMPP[1]. Un exemplu de adăugare a unui cont Google Talk este prezentat în figura 15.6.

Figura 15.6: Adăugarea unui cont de Google Talk în Pidgin

Lista de contacte este una singură, pentru toate conturile active. Aceasta facilitează și gruparea mai multor contacte într-unul singur, pentru cazul în care aceeași persoană este prezentă în lista de contacte pe mai multe conturi sau rețele.

Pidgin are inclusiv suport de chat multidirecțional, cum ar fi conferințe de Yahoo sau protocolul IRC. Aceste *chat rooms* pot fi adăugate în lista de contacte, astfel programul are o interfață uniformă.

Nucleul Pidgin este biblioteca Purple. Aceasta mai este folosită, printre altele, și de Finch, un program cu interfață text-mode, dar asemănător cu Pidgin. Cele două programe și biblioteca asociată sunt dezvoltate de echipe cu legături foarte strânse și au pornit din același cod.

Alternative la Pidgin sunt Kopete (parte a mediului KDE), Empathy (care va înlocui Pidgin în unele distribuții), sau, cu interfață text, `centerim`.

15.5 BitTorrent în Linux

Pentru protocolul BitTorrent (protocol peer-to-perr de transfer de fișiere) există mai multe aplicații în Linux. Astfel, Ubuntu vine cu **Transmission**[2] preinstalat. Simplu și ușor configurabil, programul se remarcă prin faptul că utilizatorul nu trebuie să seteze nicio preferință inițial înainte de rularea acestuia (existând definite opțiuni by-default pentru ca toate lucrurile să funcționeze perfect). Desigur, opțiunile pot fi reconfigurate și acest

[1] http://xmpp.org
[2] http://www.transmissionbt.com/

lucru se face foarte ușor din interfața grafică, nefiind necesară o experiență foarte vastă. De asemenea, clientul Transmission are cel mai mic consum de memorie dintre toate aplicațiile ce folosesc protocolul BitTorrent.

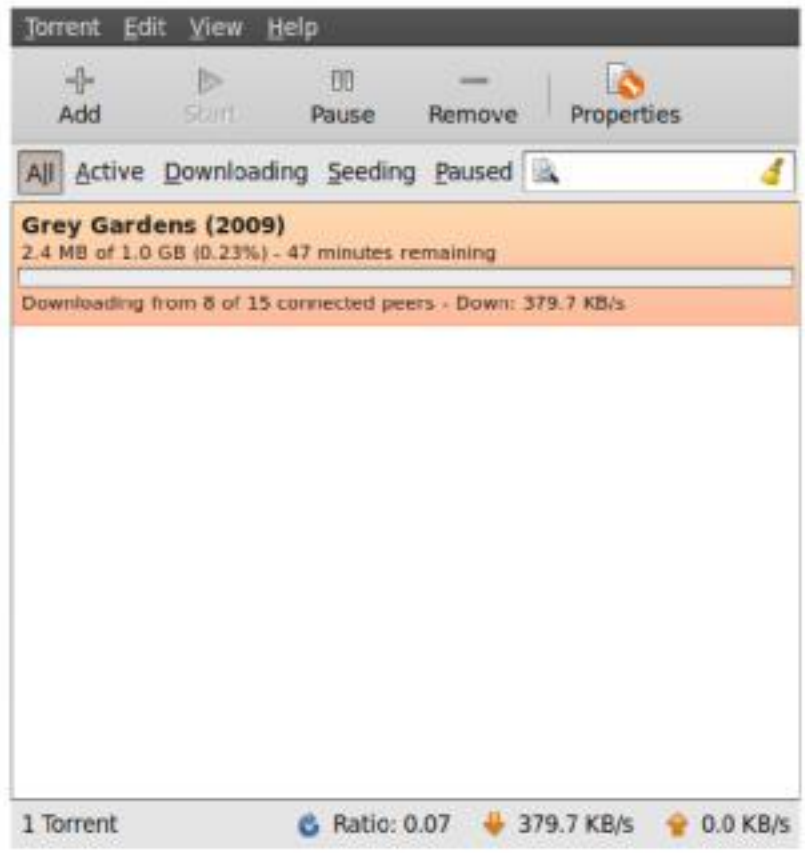


Figura 15.7: Transmission

Pentru Kubuntu există preinstalat **Ktorrent**[1]. Ca și în cazul Transmission, folosirea acestui program este foarte facilă, majoritatea configurărilor realizându-se prin interfața grafică.

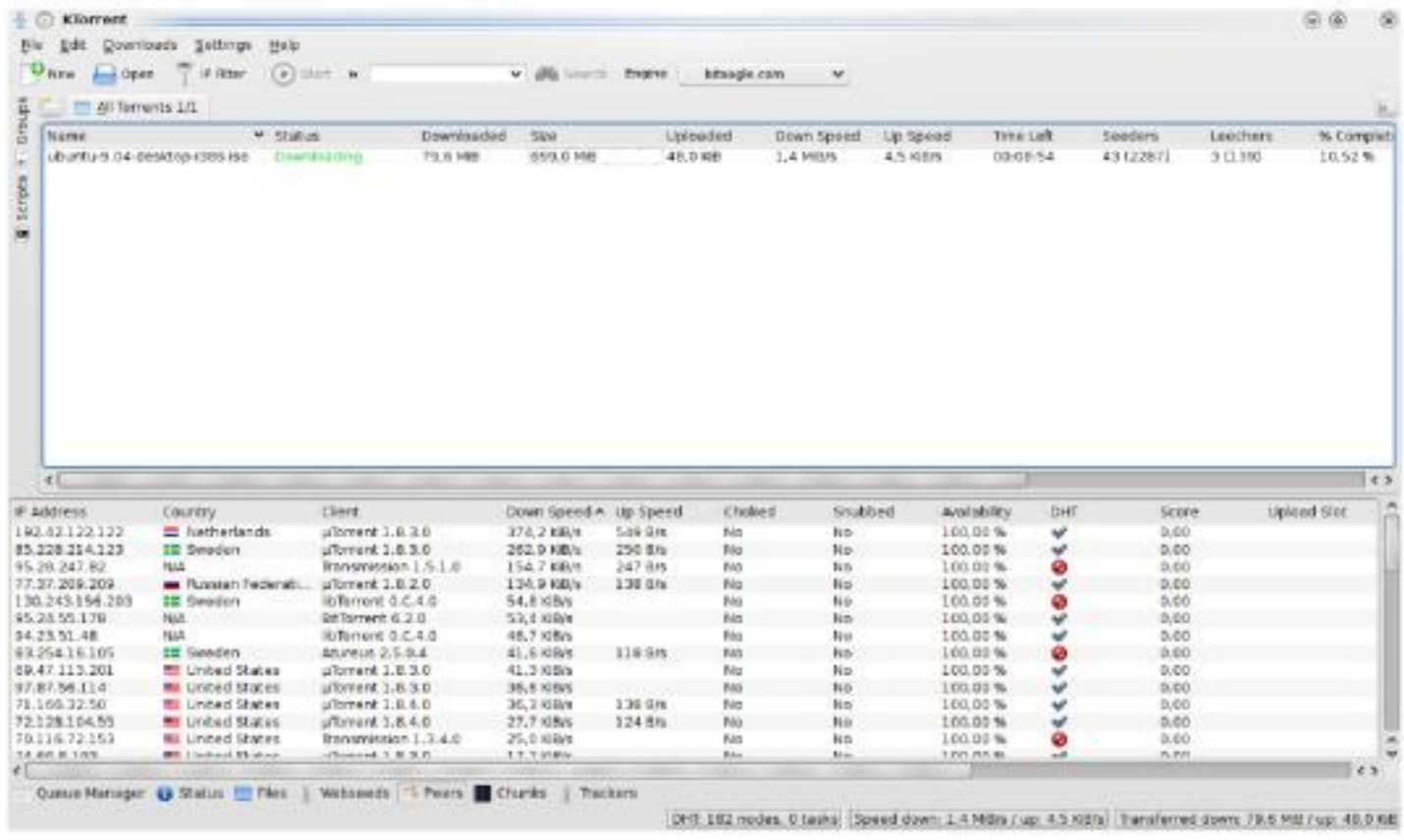


Figura 15.8: Ktorrent

Pentru cazul în care nu este disponibilă o interfață text, există posibilitatea folosirii

[1] http://ktorrent.org/

rtorrent[1]. Utilizând ncurses[2], această aplicație este foarte ușor de folosit peste o conexiune ssh sau într-un mediu fără interfață grafică.

Nu în ultimul rând, există **Limewire**[3], o aplicație cross-platform bazată pe Java, suportând facilități reduse de BitTorrent. În plus, aplicația folosește rețeaua Gnutella[4] pentru a localiza fișierele căutate.

15.6 Imprimanta în Linux

În Linux, folosirea imprimantei este condiționată de instalarea și configurarea **CUPS** (*Commmon Unix Printing System*). CUPS permite instalarea unui server care trimite solicitările către o imprimantă. Poate fi astfel folosit pentru a servi solicitări sosite din rețea. Pachetul asociat se numește `cups`.

În cadrul CUPS există pachetul `cups-pdf` care permite instalarea unei imprimante virtuale. Folosirea acestei imprimante virtuale conduce la obținerea unui fișier **PDF**[5]. În mod implicit, fișierele generate sunt stocate în directorul `PDF` din home-ul utilizatorului curent.

Imprimantele (sau serviciul CUPS) pot fi configurate în două moduri:

- prin intermediul interfeței grafice, în Gnome (vezi figura 15.9) sau KDE (vezi figura 15.10);
- prin intermediul interfeței web (vezi figura 15.11); interfața web este accesată, în mod implicit, prin intermediul URL-ului http://localhost:631/.

Figura 15.9: Gestiunea imprimantei în GNOME

Configurarea imprimantei poate fi realizată numai de un utilizator privilegiat. Operațiile uzuale sunt adăugarea imprimantei, ștergerea imprimantei, modificarea configurării de imprimantă. CUPS are instalate majoritatea driverelor de imprimantă astfel încât majoritatea imprimantelor vor putea fi configurate fără operații suplimentare. În caz de probleme sau în cazul folosirii unor imprimante ale căror drivere nu sunt prezente se recomandă accesarea site-ului OpenPrinting[6].

[1] http://libtorrent.rakshasa.no/
[2] http://www.gnu.org/software/ncurses/ncurses.html
[3] http://http://wiki.limewire.org/
[4] http://en.wikipedia.org/wiki/Gnutella
[5] http://en.wikipedia.org/wiki/Portable_Document_Format
[6] http://www.linuxfoundation.org/collaborate/workgroups/openprinting

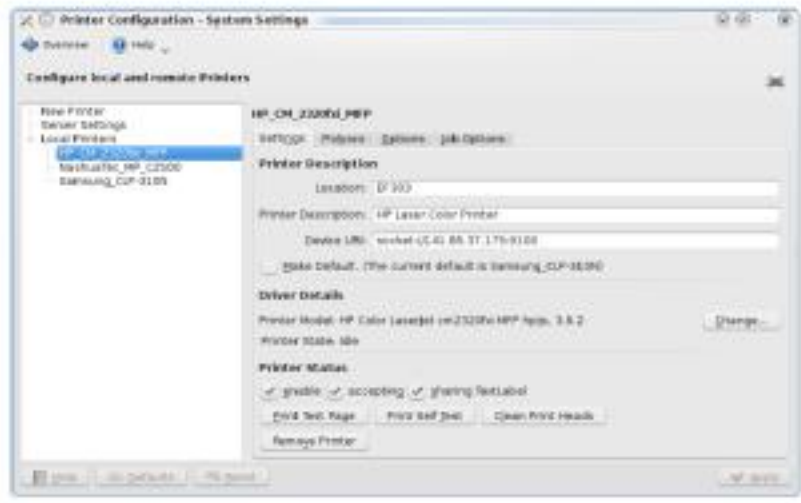


Figura 15.10: Gestiunea imprimantei în KDE

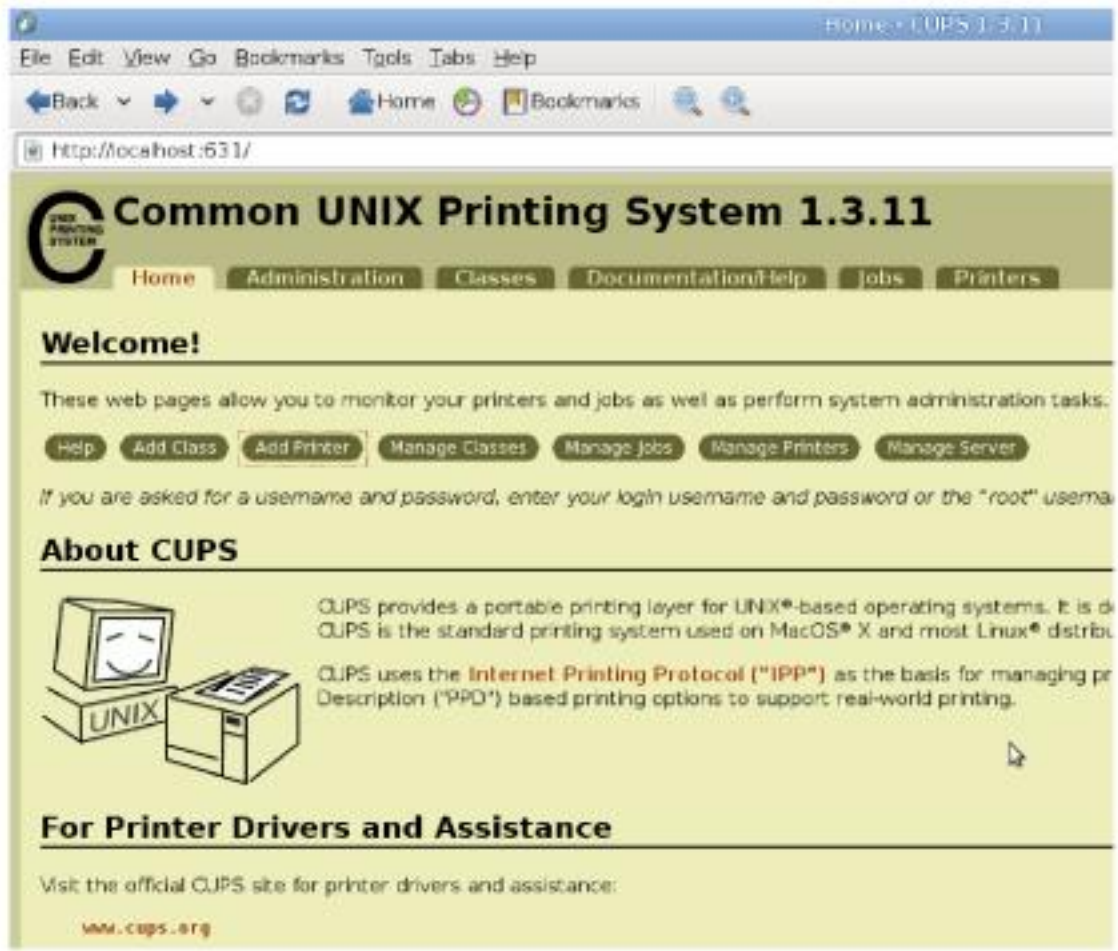


Figura 15.11: Interfața web pentru CUPS

15.7 Jocuri în Linux

Datorită originilor și mediului căruia i-a fost destinat până nu demult, Linuxul nu a reprezentat un segment de piață atractiv pentru producătorii de jocuri. De aceea nu există foarte multe jocuri care pot rula pe acest sistem de operare.

În Windows, cele mai multe jocuri folosesc pentru desenarea obiectelor 3D API-ul Direct3D de la Microsoft, parte a API-ului multimedia DirectX (DirectX a ajuns la versiunea 10 odată cu apariția Windows Vista). Pe lângă acest API (proprietar Microsoft), există și jocuri care folosesc un alt API, open source, numit OpenGL (Open Graphics Library). OpenGL este o colecție de funcții gândite a fi cross-language și cross-platform. Este printre puținele API-uri standardizate folosite în Linux pentru crearea obiectelor 3D și implicit a jocurilor 3D.

Dintre cele mai cunoscute jocuri pe Linux avem:

- Battle for Wesnoth[1] – un joc de strategie fantasy
- Nexuiz[2] – shooter, clonă Quake
- Armagetron Advanced[3] – un joc bazat pe un film SF din anii 80

Figura 15.12: Nexuiz

În afară de aceste jocuri gratis, făcute de comunități conectrate în jurul lor, există și jocuri cu suport pe Linux din partea unor firme mari. Dintre acestea probabil cea mai bine cunoscută este Id Software care oferă majoriatatea jocurilor produse și pe Linux. Epic games, productătorul seriei Unreal, oferă de asemenea pentru anumite jocuri și o variantă pentru Linux.

Wine[4] este o aplicație ce permite rularea unor programe destinate platformelor Windows pe un sistem Linux. Pentru a vedea dacă o aplicație rulează în wine și alte informații legate de rularea ei în wine puteți consulta baza de date a proiectului[5]. Winetricks[6] este un script făcut pentru instala mai ușor anumite aplicații specifice Windows, cum ar fi directx sau framework-ul .NET.

15.8 Dual-monitor în Linux

Odată cu trecerea de la monitoarele CRT la cele LCD și în același timp cu scăderea prețului la hardware, posibilitatea de a conecta două monitoare la sistemul fizic a devenit accesibilă. Creșterea productivității și a confortului în utilizarea calculatorului îndeamnă din ce în ce mai mulți utilizatori să folosească o astfel de soluție.

[1] http://www.wesnoth.org
[2] http://www.alientrap.org/nexuiz
[3] http://armagetronad.net
[4] http://www.winehq.org/
[5] http://appdb.winehq.org/
[6] http://wiki.winehq.org/winetricks

Atunci când sunt conectate două monitoare la sistem (sau un monitor și un videoproiector) acestea pot fi utilizate în două configurații:

- *mirror*, în care amândouă monitoarele arată aceeași imagine
- *extended*, în care imaginile arătate de cele două monitoare sunt diferite și alăturate

Configurația *mirror* este utilă numai în cazul în care se dorește vizualizarea aceleiași informații pe mai multe monitoare. În schimb, configurația extinsă permite mărirea spațiului de lucru, așa cum se poate vedea în figura 15.13.

Figura 15.13: Alăturarea a două desktop-uri. Fiecare este redat pe un monitor separat

Pentru realizarea unei configurații de acest gen se pot folosi utilitare grafice sau utilitare în linie de comandă. În cazul GNOME, accesând `System -> Preferences -> Display` se pot configura atât parametrii pentru fiecare din monitoare (rezoluție, rată de *refresh*) cât și activarea sau dezactivarea fiecărui monitor (figura 15.14). În plus, intrefața de configurare permite alegere modului în care sunt așezate cele două desktop-uri: unul lângă altul, unul deasupra altuia sau, spre exemplu, pe ce monitor se afișează desktop-ul din stânga.

În cazul în care se dorește realizarea unui set de configurări avansate, va trebui folosit utilitarul **`xrandr`**, descris în secțiunea 13.4.1.

Un exemplu de comandă xrandr ce permite configurarea monitorului extern pentru un laptop ca desktop extins este următoarea:

```
1 george@asgard:~$ xrandr --output VGA --mode 1920x1200 --rate 60 --pos 0x0
   --output LVDS --mode 1280x800 --rate 60 --pos 0x1200
```

Așa cum se poate vedea, pentru fiecare din monitoare (`LVDS` – ecranul laptop-ului și `VGA` – ieșirea video) se configurează trei parametri:

- rezoluția, prin parametrul `--mode`
- rata de refresh, prin parametrul `--rate` (acest parametru poate lipsi, caz în care sistemul de operare va alege automat cea mai bună variantă posibilă)
- poziția desktop-ului, prin parametrul `--pos`

Poziția desktop-ului se măsoară, la fel ca majoritatea elementelor ce țin de grafica pe calculator, începând din colțul din stânga sus, având axa `Ox` orizontală și axa `Oy`

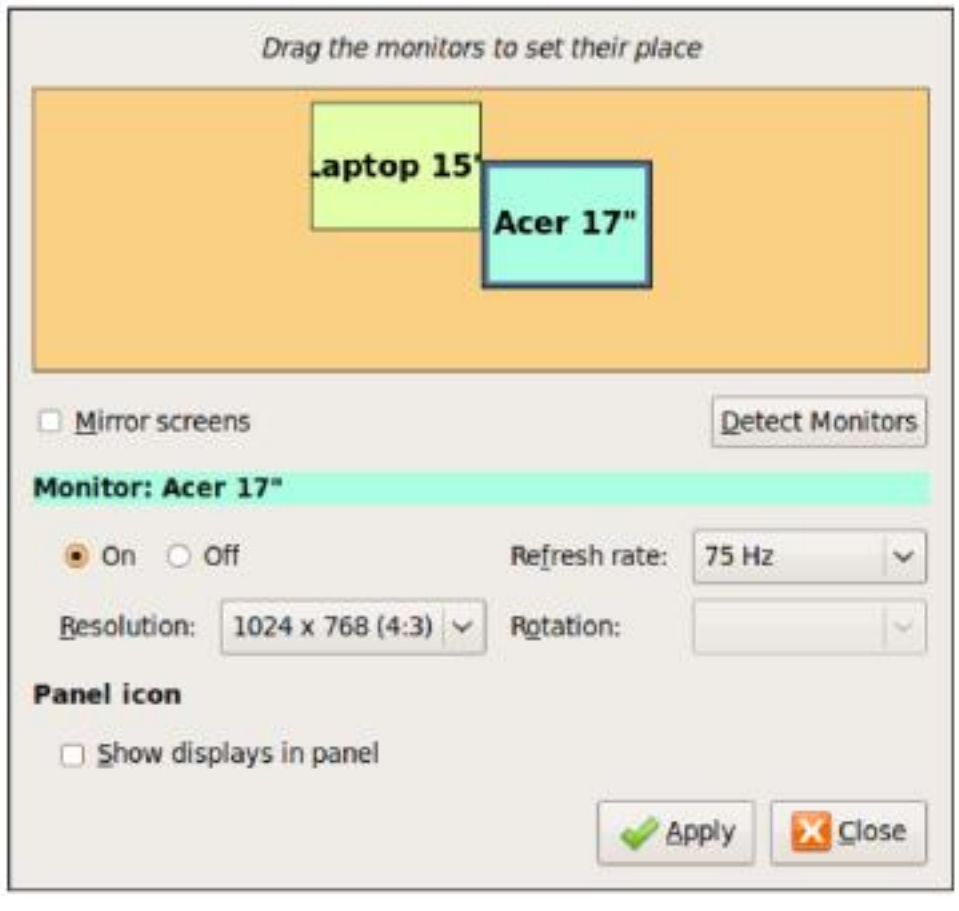


Figura 15.14: Configurări pentru desktop extins in GNOME

orientată în jos. Astfel, în exemplul de mai sus, ieșirea `VGA` se va afla cu colțul din stânga sus în origine (`X=0`, `Y=0`), iar ieșirea `LVDS` se află sub cea `VGA`, amândouă aliniate la stânga (`X=0`, `Y=1200`).

Un alt exemplu de configurare a unui desktop multiplu, care a generat configurația prezentă în figura 15.14, este:

```
1 george@asgard:~$ xrandr --output VGA --mode 1280x1024 --pos 1280x0 --
  output LVDS --mode 1280x800 --pos 0x224
```

Ieșirea `LVDS` este plasată în stânga (`X=0`), cu 224 de pixeli mai jos ca originea (`Y=224`). Ieșirea `VGA` este plasată în dreapta, la 1280 de pixeli departare de origine (`X=1280`) și la același nivel cu aceasta (`Y=0`). Cei 224 de pixeli, calculați ca diferența dintre înălțimea celor două ecrane (1024 și 800), au fost folosiți pentru a aduce partea de jos a celor două ecrane la același nivel. După cum se poate vedea și în figura 15.14, în patea de sus a ecanului mai mic (`LVDS`) apare o bandă neagră care nu va fi observată însă pe niciunul din monitoare.

15.9 Documente office în Linux

Deși suita Microsoft Office nu este disponibilă pentru Linux, interoperabilitate aproape completă cu ea se poate obține utilizând OpenOffice.

OpenOffice are, similar cu produsul Microsoft, componente pentru editarea documentelor, a foilor de calcul, gestiunea bazelor de date și realizarea prezentărilor, numite Writer, Calc, Base, respectiv Impress. În plus, există un corespondent al lui Visio numit Draw și un editor de formule matematice numit Math.

Versiunea 3.0 a OpenOffice are inclusiv suport pentru documente în format OOXML[1]. Este recomandată, în schimb, în situația în care nu este nevoie de compatibilitate cu Microsoft Office, folosirea documentelor în format nativ. Standardul se numește ODF (*Open Document Format*), este omologat ISO, și implementat de numeroase alte suite similare.

Pentru distribuția documentelor *read-only* este recomandat formatul `.pdf`. Salvarea unui document din Writer în acest format este o facilitate care vine cu OpenOffice, nefiind nevoie de instalarea unui plugin.

Desigur, există alternative la OpenOffice; KOffice este o suită similară, dar sunt și multe programe *standalone*, care pot edita un singur tip de fișier. Asemenea programe se disting prin viteză de operare superioară și necesități hardware mai reduse. Două exemple cunoscute sunt AbiWord[2] și Gnumeric[3].

Cuvinte cheie

- Rhythmbox
- Amarok
- Banshee
- codecuri
- VLC
- mplayer
- wodim (cdrecord)
- genisoimage
- GnomeBaker
- K3b
- Pidgin
- BitTorrent
- Transmission
- Ktorrent
- CUPS
- OpenOffice
- ODF

Întrebări

1. Care dintre următoarele afirmații este adevărată despre codec-urile audio și video în Linux?
   - ❑ nu există codec-uri audio/video pentru Linux
   - ❑ un utilizator trebuie să plătească pentru codec-uri în Linux
   - ❑ codec-urile sunt gratuite, dar trebuiesc compilate din surse
   - ❑ majoritatea codec-urilor sunt instalate ca dependențe ale player-elor corespunzătoare
2. Care dintre următoarele comenzi NU are legătură cu procesul de scriere a unui CD sau DVD?

---

[1] Formatul OOXML a fost introdus odată cu Microsoft Office 2007 și se distinge prin adăugarea literei "x" la extensia veche, de exemplu `.docx`

[2] http://www.abisource.com/

[3] http://projects.gnome.org/gnumeric/

- ❑ **wodim**
- ❑ **cdrecord**
- ❑ **mplayer**
- ❑ **genisoimage**

3. Ce program are și facilități pentru IRC?
   - ❑ Pidgin
   - ❑ Ktorrent
   - ❑ centerim
   - ❑ Audacious
4. CUPS este o suită de programe care se folosește la:
   - ❑ printare
   - ❑ scrierea unui CD/DVD
   - ❑ editarea de documente Office
   - ❑ CUPS este un joc
5. Pentru trimiterea unui document către un coleg pentru printare este recomandat formatul:
   - ❑ `.doc`
   - ❑ `.docx`
   - ❑ `.pdf`
   - ❑ `.odt`

Anexa A

Răspunsuri la întrebări

Capitolul 1. Introducere

1. `interpretor de comenzi` – trebuie să se poată interpreta comenzi pentru funcționarea sistemului de operare
2. `BIOS` – BIOS-ul nu este una din resursele de care este nevoie în virtualizare
3. `Larry Wall` – autorul limbajului Perl
4. `căutarea si eliminarea programelor virus` - de aceste lucruri se ocupă o aplicație antivirus
5. `OpenSolaris` – celelalte sunt sisteme de operare specializate pentru dispozitive mobile
6. `OpenVZ` – rulează doar sub Linux
7. `un proiect care produce o componentă importantă a aplicațiilor ce rulează peste nucleul Linux` - GNU's not Unix
8. `Windows` – pentru că viața e dură și marketing-ul e marketing
9. `Symbian OS` – dar iPhone OS is catching up
10. `Unix` – dezvoltat de un grup de la AT&T în anul 1969

Capitolul 2. Instalarea Linux. Configurări de bază

1. `1` – poate exista doar un root
2. `configurarea unei partiții primare` – pentru că se rulează într-un mediu temporar
3. `PalmOS`
4. `adevărat, adevărat`
5. `comanda a fost lansată de utilizatorul root` – se presupune că root are privilegii complete
6. `swap` – swap este un tip de partiție, nu un tip de sistem de fișiere

7. `virtuală` – toate celelalte sunt tipuri de partiții
8. `Ubuntu`
9. `window manager` – window manager se referă la mediul grafic
10. `adduser` – adduser este folosită pentru a adăuga utilizatori

Capitolul 3. Gestiunea pachetelor și utilizatorilor

1. `useradd` – va crea un utilizator nou folosind configurația implicită
2. `Permisiunile asupra /etc/passwd nu asigură securitate maximă.` – `/etc/passwd` poate fi citit de orice utilizator
3. `apt este un wrapper peste dpkg`
4. `legături simbolice`
5. `grupurile din care face parte` – informațiile despre grupuri sunt ținute în fișierul `/etc/group`
6. `conţine versiuni ale unor pachete diferite de ceea se găsește în repository-urile centrale` – programatorii folosesc astfel de repository-uri pentru testarea de noi versiuni ale programelor

Capitolul 4. Sisteme de fișiere

1. `suportului pentru jurnalizare` – în cazul în care se întrerupe curentul, ultimele operații pot fi refăcute pe baza jurnalului
2. `/etc/fstab`
3. `606 - 653 = rw-r-x-wx`; `a-x` va elimina dreptul de execuție de la toți (`rw-r---w-`); `u+x` va da drept de scriere owner-ului (nu se realizează schimbări); `g-r` va elimina drepturile de citire ale grupului (`rw----w-`); `o+r` va da drepturi de citire altor utilizatori; drepturile în final vor fi `rw----w-` (606)
4. `cat` – comanda **`cat`** poate fi aplicată doar fișierelor
5. `interfață de rețea de looback` – nu are corespondent în sistemul de fișiere
6. `EXT3`
7. `toți utilizatorii` – `755 = rwx-r-xr-x`
8. `afișarea de mesaje despre arhivare` – conform **`man`**
9. `mount`
10. `formatarea partiției cu sistemul de fișiere dorit` – fiind dată o partiție, pentru a putea folosi spațiul de pe aceasta, este necesară formatarea acesteia cu un sistem de fișiere

Capitolul 5. Procese

1. `init` – este primul proces pornit în sistem, cu `PID`-ul 1
2. `nohup` – comanda este folosită pentru a rula un proces cu caracteristici de daemon
3. `1` – există un singur proces `init` cu `PID`-ul 1
4. `oricâte` – nu există limită pentru procesele **`bash`** din sistem
5. `kill` – comanda doar trimite semnale proceselor
6. `fals, fals` – un program poate fi rulat de mai multe ori – astfel se vor crea mai multe procese care au aceeași imagine; **`top`** afișează pe prima coloană pid-ul
7. `pkill` – comanda permite trimiterea de semnale
8. `/proc` – un sistem de fișiere virtual, în care se găsesc informații despre fiecare proces în parte
9. `adevărat, fals` – toate PID-urile au valori pozitive; în mod normal, prioritățile au valori între -19 și +20
10. `explorer - nice` – Explorer este un shell, **`nice`** este o comandă pentru modificarea priorității unui proces

Capitolul 6. Pornirea și inițializarea sistemului

1. `BIOS, GRUB, vmlinuz, init` –
2. `un sector ce conține semnătura 0xAA55 (ultimii doi octeți)` – un sector este de tipul boot dacă el conține acești ultimi doi octeți, astfel se poate face un sector care să nu conțină un sistem de operare dar să fie bootabil având acești ultimi doi octeți
3. `upstart` –
4. `4, 1, 1, oricâte` –
5. `POST/CMOS` –
6. `stage2` –
7. `/boot/grub/menu.lst` –
8. `init, getty, login, bash` –
9. `(hd1,2)` –
10. `adevărat/adevărat` –

Capitolul 7. Analiza hardware a sistemului

1. `CPU` – CPU (Central Processing Unit) este procesorul, echivalentul unității de comandă și a unității de execuție din modelul Von Neumann

2. `dd` – **dd** copiază date *raw* de pe un dispozitiv
3. `dd` – **dd** este utilitar de copiere de date, nu de informare
4. `modinstall` – nu există comanda **`modinstall`**
5. `/dev/hda3` – /dev/hda3 reprezintă un disc
6. `/dev/ttyS0` – /dev/ttyS0 este asociat unui dispozitiv serial
7. `southbridge` – southbridge este unul din cele două chipset-uri importante de pe placa de bază
8. `TGZ` – TGZ se referă la un tip de arhivă
9. `placa grafică` – AGP (Advanced **Graphics** Port)
10. `dimensiunea registrelor` – dimensiunea registrelor este cea care denotă tipul arhitecturii (arhitectură pe 32 de biți, pe 64 de biți)

Capitolul 8. Configurări de rețea

1. `Serverul web de la adresa cs.pub.ro este oprit` – stația este online (comanda ping reușește); dacă browser-ul nu reușește înseamnă că serverul web este oprit
2. `Nu` – adresa este greșită (al treilea octet este 257; mai mare decât 255)
3. `172.16.150.0` – se face operația 172.16.150.200 ȘI-LOGIC 255.255.255.0; rezultă 172.16.150.0
4. `Dorel a generat foarte mult trafic în timp scurt, lucru specific virușilor` – opțiunea -f a comenzii ping generează foarte multe pachete (flood)
5. `Asocia nume de domenii cu adrese IP` – conform definiției
6. `102.168.124.159, pentru că a fost ultima adresă configurată de root` – masca este configurată implicit dacă nu se folosește opțiunea netmask
7. `fals, adevărat` – fișierul se cheamă /etc/network/interfaces
8. `adevărat, false` – serverele pot fi accesate; DNS este necesar pentru a asigura referirea acestora prin nume (în loc de adresă IP)
9. `calea către sursă și destinație` – conform definiției
10. `ping localhost` – toate celelalte comenzi folosesc comanda ping catre un server referit prin nume; referirea prin nume necesita un server DNS

Capitolul 9. Servicii de rețea

1. `editarea unui nou mesaj` – editarea se face pe sistemul local; nu necesită o conexiune client-server

2. `fals, fals` – telnet este protocol de nivel aplicație care funcționează peste TCP (nivel transport); comunicația este bidirecțională
3. `SSH` – SMTP și IMAP sunt protocoale de e-mail, HTTP este protocol web
4. `FTP` – FTP (File Transport Protocol) nu are nici o legătură cu serviciul web
5. `Postfix` – Firefox este browser, Outlook client de e-mail, Apache este server web
6. `pentru stabilirea conexiunii utilizatorul bestman trebuie să existe pe test.com` – comunicația FTP necesită autentificare pe server; bestman este utilizatorul care trebuie să se autentifice
7. `adevărat, fals` – din definiție
8. `Thunderbird` – Thunderbird este client de e-mail
9. `ssh` – wget și curl sunt clienți web, iar telnet permite conectarea și transmiterea de comenzi și răspunsuri HTTP
10. `niciuna dintre variante` – comunicația se realizează folosind SMTP

Capitolul 10. Elemente de securitate

1. `filtrarea pachetelor de rețea`
2. `citire și execuție` – cu **`umask 022`** fișierul va fi creat cu drepturi `644`. După execuția comenzii **`chmod go+x test`** grupul va primi drept de execuție. Într-un final, grupul va avea drepturile `r-x`
3. `_cOrN31[]sh+` – este o parolă complexă, formată cu cifre, litere și simboluri
4. `vârfuri de tensiune în rețeaua electrică` –
5. `toate variantele`
6. `fals, fals` – virusul dintr-un e-mail trebuie întâi activat; în funcție de tipul lui și de clientul de e-mail, metoda de activare poate diferi; un sistem poate fi "gazda" oricâtor viruși, atât timp cât nu este protejat
7. `numărul de clienți ai server-ului` – restul informațiilor se pot extrage direct din traficul monitorizat
8. `ulimit`
9. `definirea unei politici de securitate`
10. `/etc/shadow`

Capitolul 11. Compilare și linking

1. `ajută ca aplicațiile să fie mai simplu de întreținut` – programatorii nu mai trebuie să dezvolte și să întrețină componente, utilizând componentele disponibile în biblioteci

2. `-c`
3. `se numește portabilitate`
4. `make <target>` – fișierul `Makefile` trebuie să se găsească în directorul curent; pentru trimiterea unui fișier `Makefile` ca parametru se folosește parametrul `-f`
5. `fișiere sursă C fără directive de precompilare`
6. `static library` – codul dintr-o biblioteca statică este integrat la compilare în executabil
7. `adevărat, fals` – apelurile de sistem pot fi executate și direct de către aplicații
8. `fișierul este executabil` – restul informațiilor reies din ieșirea comenzii **`file`**
9. `fals, fals` – programele interpretate se depanează mai ușor decât cele compilate deoarece procesul de dezvoltare nu mai presupune compilarea codului; în urma compilării nu rezultă fișiere în limbaj de asamblare (fișierele în limbaj de asamblare nu se pot executa direct pe procesor)
10. `strace <program>`

Capitolul 12. Shell scripting

1. `Flash` – *Flash* este un plug-in pentru web browser
2. `grep` – este singura comandă dintre cele prezentate care poate primi ca parametru o expresie regulată
3. `<<` – este operatorul pentru deschiderea unui *here document*
4. `test a ==b` – `[$a == $b]` nu are operanzii separați de paranteze; celelalte opțiuni nu sunt folosite pentru testarea egalității
5. `Sortează liniile în format numeric`
6. `cut` – **`cut`** este folosit pentru extragerea de componente dintr-o linie de text
7. `Viteza mare de execuție` – scripturile shell trebuie să fie interpretate, motiv pentru care au o viteză mică de execuție
8. `Prezintă calea către utilitarul folosit pentru rularea scriptului`
9. `repeat` – nu există comanda `repeat` în Bash
10. `find . -mindepth 2 -type f` – comanda afișează recursiv doar fișierele din subdirectoare

Capitolul 13. Mediul grafic

1. `PNG` – PNG este un format bitmap cu suport pentru transparență (*alpha channel*)

2. `UTF-8` – `UTF-8` este un standard de codificare a caracterelor în fișiere
3. `permite unui utilizator să se autentifice în interfața grafcă` – conform definiției
4. `Xorg și bitmap` – Xorg este o implementare de server X, bitmap este un tip de imagini
5. `Dorel nu are dreptul să pornească serviciul kdm` – din prompt se poate observa că utilizatorul nu este autentificat ca `root`
6. `Spotlight` – Spotlight este un serviciu de indexare și căutare a fișierelor în Mac OS X
7. `KDE` – KDE este un *Desktop Environment*
8. `System Information` – restul aplicațiilor sunt folosite pentru a afla informații în linia de comandă
9. `Adevărat, Fals` – Un Window Manager controlează cum arată ferestrele aplicațiilor prin intermediul componentei *window decorator*. Clientul X nu interacționează cu utilizatorul direct; server-ul X face acest lucru.
10. `Xorg` – Xorg este o implementare de server X

Capitolul 14. Utilitare pentru dezvoltare

1. `Vim` – **`gdb`** este un debugger, **`ctags`** este un program utilizat pentru a prelucra o sursă și scoate un fișier cu indecși și **`astyle`** este un *code beautifier*
2. `Valgrind` – **`git`** este un program de *revision control*, **`indent`** este un *code beautifier* și Trac este un utilitar de management a proiectelor
3. `boot` – `checkout/clone` aduc local o copie a repository-ului, `commit` adăugă modificările la repository și `update` aduce repository-ul local la cea mai nouă versiune
4. `licențierea codului sub GPL` – o licență nu va influența frumusețea sau lizibilitatea codului.
5. `OpenOffice` – OpenOffice este Office Suite, nu oferă unelte necesare dezvoltării codului.
6. `ctags` – **`splint`** este un utilitar de analiză statică a codului, **`indent`** este un code *beautifier* și **`gdb`** este un utilitar de depanare
7. `graphic` – în modul `visual` se intră apăsând tasta *v*, în modul `command` se intră apăsând :, în modul `insert` se intră apăsând tasta *i*
8. `:w` – *:q* închide editorul, *:s* va face replacei, *:help* lansează într-un tab fișierul de help
9. `git` – Vim este un editor, Eclipse este un IDE și ddd este un debugger
10. `Trac` – GTK este un *widget toolkit*, ddd este un *debugger* și Waterfall este un model de a dezvoltare software-ului

Capitolul 15. Viața în Linux

1. `Majoritatea codec-urilor sunt instalate ca dependenţe ale player-elor corespunzătoare.` – pentru o utilizare mai ușoară (nu trebuie instalare ulterior codecurile)
2. `mplayer` – este un media player
3. `Pidgin` – `libpurple`, biblioteca utilizată de Pidgin are suport și pentru `IRC`
4. `printare` – *Common Unix Printing System*
5. `.pdf` – celelalte formate nu sunt suportate integral pe toate sistemele

Bibliografie

[1] Tom Adelstein and Bill Lubanovic. *Linux System Administration*. O'Reilly Media, March 2007.

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *compilers: Principles, Techniques, and Tools*. Addison Wesley, January 1986.

[3] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, second edition, April 2008.

[4] Nitesh Dhanjani and Justin Clarke. *Network Security Tools*. O'Reilly Media, April 2005.

[5] Bruce Eckel. *Thinking in Java*. Prentice Hall PTR, fourth edition, February 2006.

[6] John Fusco. *The Linux Programmer's Toolbox*. Prentice Hall, March 2007.

[7] Simson Garfinkel, Gene Spafford, and Alan Schwartz. *Practical Unix and Internet Security*. O'Reilly Media, third edition, February 2003.

[8] Mark G. Graff and Kenneth R. Van Wyk. *Secure Coding: Principles and Practices*. O'Reilly, July 2003.

[9] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, fourth edition, September 2006.

[10] Michael Horton and Clinton Mugge. *Hacknotes - Network Security Portable Reference*. McGraw-Hill Osborne Media, July 2003.

[11] Steve Hunger. *Debian GNU/Linux Bible*. Wiley, May 2001.

[12] Brian W. Kernighan and Rob Pike. *The Practice of Programming*. Addison-Wesley Professional, February 1999.

[13] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, second edition, April 1988.

[14] Donald E. Knuth. *Art of Computer Programming*. Addison-Wesley Professional, October 1998.

[15] John R. Levine. *Linkers and Loaders*. Morgan Kaufmann, October 1999.

[16] Robert Love. *Linux Kernel Development*. Novell Press, second edition, January 2005.

[17] Robert Love. *Linux System Programming*. O'Reilly Media, September 2007.

[18] Neil Matthew and Richard Stones. *Beginning Linux Programming*. Wrox, November 2007.

[19] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, fourth edition, November 2008.

[20] Cyrus Peikari and Anton Chuvakin. *Security Warrior*. O'Reilly Media, January 2004.

[21] Eric S. Raymond. *The Art of Unix Programming*. Addison-Wesley Professional, October 2003.

[22] Răzvan Rughiniș, Răzvan Deaconescu, Andrei Ciorba, and Bogdan Doinea. *Rețele locale*. Printech, September 2008.

[23] Răzvan Rughiniș, Răzvan Deaconescu, George Milescu, and Mircea Bardac. *Utilizarea sistemelor de operare*. Printech, September 2007.

[24] Joel Scambray, Mike Shema, and Caleb Sima. *Web Applications (Hacking Exposed)*. McGraw-Hill Osborne Media, second edition, June 2006.

[25] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, second edition, October 1996.

[26] Avi Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. John Wiley & Sons, seventh edition, December 2004.

[27] W. Richard Stevens. *Advanced Programming in the Unix Environment*. Addison-Wesley Professional, June 1992.

[28] W. Richard Stevens. *TCP/IP Illustrated*. Addison-Wesley Professional, January 1994.

[29] W. Richard Stevens. *Unix Network Programming*. Prentice Hall, second edition, January 1998.

[30] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, fourth edition, August 2002.

[31] Andrew S. Tanenbaum. *Structured Computer Organization*. Prentice Hall, fifth edition, June 2005.

[32] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, third edition, December 2007.

[33] Larry Wall, Tom Christiansen, and Jon Orwant. *Programming Perl*. O'Reilly Media, third edition, July 2000.

Glosar

-Wall, 326
-- help, 43
--help, 372
.vimrc, 477
/dev, 201
/etc/hostname, 236
/etc/hosts, 235
/etc/network/interfaces, 225
/etc/resolv.conf, 233
/proc/cpuinfo, 208
/proc/modules, 194
înlănțuirea comenzilor, 382
întrerupere, 204
__cplusplus, 361

ACL (*Access Control List*), 88
ACPI (*Advanced Configuration and Power Interface*), 198
adresă IP, 219
adresă IPv6, 219
adresă MAC, 219
agile, 497
Anjuta, 469, 494
apel de sistem, 186
apeluri de sistem, 487
API (*Application Programming Interface*), 7
aplicații de bază, 2
apropos, 43, 372
arhitectura unui sistem de calcul, 188
arhivare, 82
asamblare, 328, 333
astyle, 472
autocompletion (bash), 369
AVFS (*Anti-Virus Filesystem*), 103
awk, 410

background, 130
bash, 366
bg, 132
bibliotecă, 341
bibliotecă partajată, 342, 486
bibliotecă statică, 342
biblioteci cu legare dinamică, 342
biblioteci cu legare statică, 342
BIOS (*Basic Input Output System*), 153
bitmap, 442
booting, 152
bootloader, 154
branch, 479
bug, 489, 497
bug-report, 497

C si C++, 357
Căsuță poștală, 264
cache L1, 189
cache L2, 189
cachegrind, 492
cale absolută, 72
cale relativă, 72
case (bash), 367, 394
cat, 378, 400
certificat digital, 311
chainloading, 159
checkout, 479
chmod, 291
chown, 291
chroot, 300
CLI (*Command Line Interface*), 38, 366
client, 254
cmdlets, 434
CMOS (*Complementary Metal Oxide Semiconductor*), 153
code folding, 473
coding style, 470
comandă, 42
commit, 479
compilare, 321, 328, 331
compilare din surse multiple, 327
compilator, 321, 324

configurare bash, 426
configurare prompt shell, 429
configurare vim, 477
configure, 484
consolă, 40
consola GRUB, 164
controlul accesului, 281
controlul versiunii, 479
criptare, 310
csh, 366
ctags, 478, 482
CTRL-\, 139
CTRL-ALT-DEL, 142
CTRL-C, 139
CTRL-Z, 131, 139
cut, 403
Cygwin, 356

daemon, 133
darcs, 479
dd, 206
DDD (*Data Display Debugger*), 489, 491
Debian, 25
debug, 489
debugger, 489
depanare, 489
depmod, 195
desktop environment, 447
dezasamblare, 334
dhclient, 225
DHCP (*Dynamic Host Control Protocol*), 251
director, 69, 74
display manager, 447
dispozitiv bloc, 203
dispozitiv caracter, 203
dispozitive de intrare/ieșire, 187
distribuție Linux, 24
DLL (*Dynamic-link library*), 342
DMA, 206
DNS (*Domain Name System*), 223, 251
documentație html, 499
documentație LaTeX, 499
DoS (*Denial of Service*), 281
Doxygen, 499
driver, 3, 193

echo, 377
Eclipse, 469, 494
ELF (*Executable and Linking Format*), 340
Emacs, 320
emulare, 17
emulator de terminal, 41
EncFS, 103
Ethernet, 252
ethtool, 239
exit, 377
export, 377
expresie regulată, 407
extern, 361
Extreme Programming, 497

feature creep, 282
fg, 132
fișier, 69, 74
fișier executabil, 339
fișier obiect, 333
find, 416
Firewall, 310
firewall, 310
for (bash), 367, 395
foreground, 130
fork bomb, 281
frecvență, 188
fsck, 212
FTP (*File Transfer Protocol*), 251
funcții bash, 424
FUSE, 103

gateway, 222
GCC (*GNU Compiler Collection*), 325
gcc în Windows, 356
GDB (*GNU Debugger*), 489
GID (Group Identifier), 55
git, 479, 481
GNU, 23
GNU Autotools, 485
GNU CPP (*GNU C preprocessor*), 331
GNU/Linux, 23
GParted, 49
grep, 405, 407
GRUB, 162
GUI, 7
GUI (*Graphical User Interface*), 39
GUI (grafical user interface), 366

hard-disk, 190
hardware, 186

head, 402
helgrind, 492
here document, 380
here string, 382
hipervizor, 16
host, 234
hostname, 236
HTML (*Hypertext Markup Language*), 270
HTTP (*Hypertext Transfer Protocol*), 251, 270
hwinfo, 198

ICMP, 236
IDE (*Integrated Development Environment*), 320, 469, 494
IDS (*Intrusion Detection System*), 312
if (bash), 367, 391
ifconfig, 225
imagine de CD, 212
imagine vectorială, 443
IMAP (*Internet Mail Access Protocol*), 251, 264
indent, 471
info, 43, 372
init, 117, 143
insmod, 195
instalarea din surse, 484
interfață de rețea, 225
interpretare, 321
interpretor, 321
IP, 218
ipconfig, 240
IRC (*Internet Relay Chat*), 501
istoric (bash), 369

Javadoc, 499
jeton hardware, 288
jobs, 132
jurnalizare, 93

kernel, 2, 185
kill, 137
killall, 138
kswapd, 143

LAN (*Local Area Network*), 218
ld, 486
ldconfig, 486
ldd, 486
legătură simbolică, 74
libnotify, 460
limbaj asamblare, 332
limbaje de nivel înalt, 322
limbaje de nivel scăzut, 322
limbaje de programare de nivel înalt, 322
limbaje de programare de nivel scăzut, 322
link-editare, 328, 337
linker, 338
Linux, 23
Live CD, 31
locate, 415
loopback, 226
lsmod, 194
lspci, 197
lsusb, 196

mașină virtuală, 16
make, 345
man, 43, 372
MAN (*Metropolitan Area Network*), 218
management proiecte software, 496
massif, 492
Master Boot Record, 31
MBR (*Master Boot Record*), 156, 211
medii integrate de dezvoltare, 494
memcheck, 492
memorie, 187
memorie cache, 189
memorie virtuală, 285
MinGW (*Minimalist GNU for Windows*), 356
modprobe, 195
module de kernel, 194
montarea unui sistem de fișiere, 97
MSVC (*Microsoft Visual C++*), 469, 502
multi-core, 188
multiprogramare, 116
multitasking, 116

name-mangling, 360
nice, 146
nl, 400
nm, 335
nohup, 134
northbridge, 191
NTFS-3G, 103
NTLoader, 178

objdump, 334

one liner, 378
operatorul &, 130
operatorul $((...)), 420
operatorul >>, 380
operatorul <<<, 382
operatorul <<, 380
operatorul |, 140, 382
optimizare, 335
optimizarea compilării, 335

pachet software, 58
pachete necesare pentru dezvoltarea codului C, 324
pager, 77
paginator, 77
parametrii unui script shell, 421
partiți logică, 31
partiție, 30
partiție de swap, 30
partiție extinsă, 31
partiție primară, 31
PATH, 74
pdflush, 143
peer to peer, 254
pgrep, 123
PID, 113
ping, 236
PKI (*Public Key Infrastructure*), 287
pkill, 138
placa de bază, 191
Poșta electronică, 263
poff, 245
pon, 245
POP3 (*Post Office Protocol*), 251, 264
port, 255
port I/O, 205
portabilitate, 354
POSIX (*Portable Operating System Interface*), 356
POST (*Power-on Self Test*), 154
PPP, 243
PPPoE, 243
pppoeconf, 243
preprocesare, 328, 330
principiul celui mai mic privilegiu, 281
principiul limitării drepturilor, 281
printf, 377
proces, 112
proces de dezvoltare software, 496
procesor, 188
procfs, 114, 127
program, 112
prompt, 42
protocol, 249
ps, 117

RAM, 190
ramdisk, 32
raster, 442
rețea de calculatoare, 217
read, 378
readline, 368
Redmine, 497
registre, 188, 189
registre vim, 476
repository, 479, 497
resolvconf, 233
reverse search (bash), 369
rmmod, 195
roadmap, 497
root, 45
ruter, 219, 252

S.M.A.R.T., 200
scanarea porturilor, 307
scheduling, 115
schimbare de context, 116
sector de boot, 154
sed, 408
Semantic Web, 268
semnal, 135
semnale, 487
seq, 396
server, 254
set, 377
setgid, 299
setuid, 299
sfdisk, 211
shebang, 375
shell, 41, 365
shell script, 365
shift, 422
shutdown, 44
SIGKILL, 135
SIGSEGV, 135
SIGTERM, 135
simbolul $, 419
sistem de fișiere, 70
sistem de operare, 1
sistem multi-core, 115

sisteme centralizate de versionare, 479
sisteme distribuite de versionare, 479
sisteme multi-tasking, 5
sisteme multi-user, 5
smartmontools, 200
SMTP (*Simple Mail Transfer Protocol*), 251, 265
SNMP (*Simple Network Management Protocol*), 251
SO (sistem de operare), 1
southbridge, 191
spațiu kernel, 186
spațiu utilizator, 186
spam, 267
splint, 482
sshfs, 103
starea unui proces, 115
stivă de protocoale, 249
strace, 488
su, 46
Subversion, 479, 480
sudo, 47, 301
SVN (Subversion), 479, 480
swap, 30, 213
swapping, 30, 141
switch, 252
syntax highlighting, 473
sysctl, 210
sysfs, 196
system tray, 460

tabela de partiții, 157
tac, 400
tag, 479
tail, 402
Task manager, 142
TCP (*Transmission Control Protocol*), 251
terminal virtual, 40
test (bash), 391
TFTP, 254
time, 126
top, 124
tr, 404
Trac, 497
traceroute, 237
troian, 306
trunchiere fișier, 382
TTL, 237
TUI (*Text User Interface*), 7, 39

Ubuntu, 25
udev, 201
UDP(*User Datagram Protocol*), 251
UID (User Identifier), 55
ulimit, 302
umask, 292
Unicode, 444
unitatea aritmetică și logică, 187
unitatea de comandă, 187
update, 479
Usenet, 501
utilizare biblioteci, 344

valgrind, 492
variabila `$?`, 392
variabila `$@`, 422
variabila `$#`, 421, 422
variabile make, 352
variabile speciale shell, 423
VBR (*Volume Boot Record*), 156
VCS (Version Control System), 479
vectorial, 443
vi, 320
vim, 320, 473
vimtutor, 474
virtualizare, 16
virus, 305
VNC (Virtual Network Computing), 462

WAN (*Wide Area Network*), 218
waterfall, 497
wc, 405
webmail, 266
whereis, 372
while (bash), 367, 398
wiki, 497
window decorator, 447
window manager, 446
WLAN, 253
working copy, 479
worm, 306

X11, 444
XHTML, 271

ISBN 978-606-521-386-9